NETWORK-STRUCTURED ERROR FLATTENING FOR POWER GRIDS AND OTHER REAL-WORLD NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Colin Vincent Ponce

May 2016

© 2016 Colin Vincent Ponce
ALL RIGHTS RESERVED

NETWORK-STRUCTURED ERROR FLATTENING FOR POWER GRIDS AND OTHER REAL-WORLD NETWORKS

Colin Vincent Ponce, Ph.D.

Cornell University 2016

From power grids to social networks to neuroscience, networks are increasingly important in science today. They are, however, inherently hard to study. On one hand, phenomena beginning in one part of a network can have complex and global effects on the rest of the network, and so behavior is frequently difficult to predict without simulations. On the other hand, modern networks are often massive, containing hundreds of millions or even billions of nodes. Due to this, network computations often require specialized algorithms that exploit network structure to perform their tasks efficiently.

In this work, we study matrix-based network computations and the relationship between network structure and linear algorithms. Out algorithms use either low rank upates or coarse grid projections to transform the problem into a smaller one that is exactly or approximately equivalent to the original. We refer to these techniques as error flattening methods.

We present three examples: a method for fast detection and identification of power grid topology errors; a nonlinear multigrid method to solve the power flow equations; and a two-part iterative method to solve graph Laplacian systems.

BIOGRAPHICAL SKETCH

Colin Ponce was born May 12, 1988 in Hartford, Connecticut. He graduated magna cum laude from Princeton University with a BSE in Computer Science. He won the National Defense Science and Engineering Graduate Fellowship in 2012, and received a Master of Science in Computer Science from Cornell University in 2014.

I dedicate this to my parents.

ACKNOWLEDGEMENTS

I would like to thank Panayot Vassilevski, with whom I have worked on multigrid methods. I would also like to thank Ken Birman and Lang Tong for their collaborations on power grid work, and Martin Wells for his statistical knowledge and teaching. Finally, I would like to thank my adviser David Bindel, who has played a central role in my education at Cornell University and in my development as a researcher.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

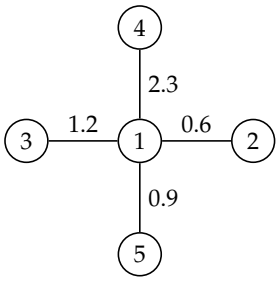
INTRODUCTION

1.1 Computational Network Science

Networks are not new, but our ability to study them on a large scale is. The study of networks perhaps began with Jacob Moreno, who in 1933 developed the sociogram, a graphical representation of social interactions [51]. In the 1950's, Paul Erdős and Alfréd Rényi introduced the concept of a random graph, as well as the Erdős-Rényi random graph model [22]. Since then, computer technology has enabled modern network science.

As computer technology has progressed, however, so has the size of networks we wish to study. While Moreno originally studied social networks of a few dozen people, today we are interested in studying networks with hundreds of million or even billions of nodes. For example, the Facebook network has approximately 1.5 billion people, and the human brain has approximately 86 billion neurons [1, 8]. Fortunately, virtually all large networks of interest today are sparse, and so the average node degree of the resulting graphs is typically quite small. Because of this, algorithms that exploit sparsity patterns can be highly effective, making it practical to compute on such large networks.

In this work, we focus on sparse numerical linear algebra for network computations. Network problems that reduce to matrix computations are common, as one may view many mathematical models of networks as network-structured systems of linear or quadratic functions. In addition, sparse matrix methods already exist for many problems in numerical linear algebra, and so it is a field



$$\begin{bmatrix} 0 & 0.6 & 1.2 & 2.3 & 0.9 \\ 0.6 & 0 & 0 & 0 & 0 \\ 1.2 & 0 & 0 & 0 & 0 \\ 2.3 & 0 & 0 & 0 & 0 \\ 0.9 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Adjacency matrix

$$\begin{bmatrix} 6 & -0.6 & -1.2 & -2.3 & -0.9 \\ -0.6 & 0.6 & 0 & 0 & 0 \\ -1.2 & 0 & 1.2 & 0 & 0 \\ -2.3 & 0 & 0 & 2.3 & 0 \\ -0.9 & 0 & 0 & 0 & 0.9 \end{bmatrix}$$

Laplacian matrix

Figure 1.1: A small graph and its associated adjacency and graph Laplacian matrices.

with the tools necessary to develop algorithms for large, sparse networks.

1.2 Mathematical Framework and Operator Contraction

Networks are often represented as graphs, and graphs as matrices. Several matrix representations are common, but all of them follow some general patterns. If u is the index of a graph node, row and column u correspond to node u, and element A_{uv} is nonzero if there is an edge between nodes u and v. If the graph is undirected, then A has a symmetric nonzero structure, while if it is directed A is in general nonsymmetric. A sparse network leads to a sparse matrix.

The values of the nonzero elements of A, and the values of the diagonal, depend on the particular matrix representation. Two standard representations are the *adjacency matrix* and the *graph Laplacian matrix*. If A is an adjacency matrix, then it has a zero diagonal, and the value of A_{uv} is simply the weight of the edge between nodes u and v (or 1 if the graph is unweighted). The graph Laplacian matrix is $L = D - A$, where D is a diagonal matrix of row sums of A. An example of these matrices is shown in Figure 1.1.

In this work we focus on computations with the graph Laplacian matrix.

This matrix is used in many application areas. Its name comes from the fact that, in a grid graph, the graph Laplacian is the discretization of the second-order differential Laplace operator. In n-dimensional space, the Laplace operator is the second-order differential operator

$$\Delta = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2}. \tag{1.1}$$

Applying this operator to a function f produces the function

$$\begin{aligned} \Delta f(x_1, ..., x_n) &= \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2} f \\ &= \lim_{h \to 0} h^{-2} \sum_{i=1}^{n} -f(..., x_i + h, ...) + 2f(..., x_i, ...) - f(..., x_i - h, ...). \end{aligned} \tag{1.2}$$

One can therefore interpret the graph Laplacian matrix as the discretization L obtained from the Laplace operator by choosing a constant distance $h > 0$. If the vector $\boldsymbol{f}$ is the vector containing the values of the function f at the various grid points, then (abusing notation slightly)

$$\Delta f \approx L\boldsymbol{f}.$$

For this reason, the graph Laplacian is frequently used when computing the numerical solutions to second-order elliptic differential equations. It is used in a variety of other contexts as well, however. For example, *spectral partitioning* uses the eigenvector associated with the second-smallest eigenvalue of L to approximate the minimum balanced cut of a graph (see, e.g. [28]). It is also used to study electrical networks, as Kirchoff's Law shows that, if $\boldsymbol{v}$ is a vector of nodal voltages around a network, then $L\boldsymbol{v}$ is a vector of current injections around a network (see, e.g. [30]).

A fair amount of research effort has been put into solving linear systems involving graph Laplacians. These generally fall in one of two categories:

1. **Sparse direct methods.** These methods are usually not about graph Laplacians in particular, but more rather about matrices with a particular sparse nonzero structure. They carry out a finite number of operations in order to solve the problem exactly (if using exact arithmetic). These methods involve a setup phase in which the matrix is factored, perhaps using LU or Cholesky factorization, followed by a much cheaper solution phase (see, e.g. [20]).
2. **Iterative methods.** These methods take a guess $\boldsymbol{x}_k$ and produce a new guess $\boldsymbol{x}_{k+1}$ that is closer to the solution. The process repeats until some desired accuracy is achieved (see, e.g. [58]). While the speed of the iteration step may depend on the nonzero structure of the matrix, the convergence rate of the method depends on the spectral properties of the matrix. The graph Laplacian matrix L is positive semi-definite, making it well-suited to many iterative methods. The adjacency matrix A, however, is indefinite, and so is not well-suited to many iterative methods.

As networks grow, so do their Laplacians. Error flattening techniques help us to solve huge, sparse systems of equations by compressing expensive matrix calculations into operations on smaller matrices. They perform two steps:

1. Perform a partial solve so that the remaining error lies (either exactly or approximately) in some known, low-dimensional subspace.
2. Construct a small matrix representing just this low dimensional subspace, and (either exactly or approximately) solve that smaller problem.

An exact solve results in a direct solution method, whereas an approximate solve is usually used in an iteration.

Next, we introduce two specific of error flattening techniques, one exact and one approximate.

1.3 Matrix Augmentation

Suppose we have a block matrix of the form

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \in \mathbb{R}^{n+k \times n+k}, \tag{1.3}$$

where A is $n \times n$, B is $n \times k$, C is $k \times n$, and D is $k \times k$, and $k \ll n$. Suppose we want to solve the linear system $M\boldsymbol{z} = \boldsymbol{b}$, where

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \in \mathbb{R}^{n+k} \qquad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} \in \mathbb{R}^{n+k}$$

Assume that k is small enough that dense matrix solves are fast, A is invertible, and that we have a fast algorithm for solving linear systems involving A.

Block Gaussian elimination is a standard solution method in this context. If we premultiply the first block row by the matrix CA^{-1} and subtract it from the second block row, we obtain the equation

$$\left(D - CA^{-1}B\right)\boldsymbol{y} = \boldsymbol{d} - CA^{-1}\boldsymbol{c}. \tag{1.4}$$

The matrix on the left-hand side is called the *Schur complement* of A in M. The construction of this linear system requires $k + 1$ solves with the matrix A. However, once it is complete, it is of size k, and so can be solved quickly. Solving for $\boldsymbol{y}$ and substituting back into the original system gives

$$A\boldsymbol{x} = \boldsymbol{c} - B\boldsymbol{y} = \boldsymbol{c} - B\left(D - CA^{-1}B\right)^{-1}\left(\boldsymbol{d} - CA^{-1}\boldsymbol{c}\right). \tag{1.5}$$

As we have a fast method for A by assumption, this system can also be solved quickly. Thus, Equations (1.4) and (1.5) provide a two-step method for solving this linear system.

Note that k of the solves with A were used to construct the Schur complement. If M is going to be used for multiple solves, this computation does not need to be repeated, and so subsequent solves only require one solve with A and one solve with the size-k Schur complement.

1.3.1 The Sherman-Morrison-Woodbury Formula

A special case of matrix augmentation gives rise to an important formula in numerical linear algebra:

Theorem 1. *(The Sherman-Morrison-Woodbury Formula) Suppose that A and C are nonsingular $n \times n$ and $k \times k$ matrices respectively, and U and V are size $n \times k$. Let*

$$\hat{A} = A + UCV^T.$$

If the $k \times k$ matrix $C^{-1} + V^T A^{-1} U$ is invertible, then

$$\hat{A}^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + V^T A^{-1} U\right)^{-1} V^T A^{-1}. \tag{1.6}$$

Proof. We prove this by showing that the linear system $\hat{A}\boldsymbol{x} = \boldsymbol{b}$ can be solved using Equation (1.6) for any $\boldsymbol{b}$.

Construct the extended linear system

$$\begin{bmatrix} A & U \\ V^T & -C^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}.$$

On the one hand, using the bottom row, we see that

$$V^T \boldsymbol{x} = C^{-1} \boldsymbol{y} \implies \boldsymbol{y} = C V^T \boldsymbol{x}.$$

Substituting this back into the first row and inverting gives us the statement

$$\boldsymbol{x} = \hat{A}^{-1} \boldsymbol{b}.$$

On the other hand, Equation (1.5) shows us that

$$\boldsymbol{x} = A^{-1} \boldsymbol{b} - A^{-1} U \left(C^{-1} + V^T A^{-1} U \right)^{-1} V^T A^{-1} \boldsymbol{b}.$$

□

So, we see that one can use matrix augmentation and the Sherman-Morrison-Woodbury formula to contract key parts of linear operators into small matrices, allowing for efficient solution methods.

We use matrix augmentation in Chapter 2 to solve multiple linear systems of the form in Equation (1.3) where B, C, and D change but A remains the same. By factoring A ahead of time, error flattening allows us to solve those systems quickly. We also apply the Sherman-Morrison-Woodbury formula recursively in Chapter 4 to accelerate the solution of linear systems with a particular topology.

1.4 Coarse-Grid Projections

Suppose we wish to solve a linear system $A\boldsymbol{x} = \boldsymbol{b}$, and we know that $\boldsymbol{x}$ lies near some subspace spanned by the orthogonal matrix $P \in \mathbb{R}^{n \times k}$ that we call an *interpolation matrix*; that is, $\boldsymbol{x} \approx P\boldsymbol{x}_c$ for some $\boldsymbol{x}_c \in \mathbb{R}^k$. Then to approximate the

solution, we wish to solve the system

$$AP\boldsymbol{x}_c = \boldsymbol{b}.$$

However, this is an overdetermined system, so unless $\boldsymbol{x} = P\boldsymbol{x}_c$ exactly, no solution exists. Instead, we approximate the solution by solving the least-squares problem

$$\min_{\boldsymbol{x}_c} ||AP\boldsymbol{x}_c - \boldsymbol{b}||_N^2,$$

where $N \in \mathbb{R}^{n\times n}$ is a symmetric positive-definite matrix that defines an inner-product norm. A calculation shows that, if $N = RR^T$ for some matrix $R \in \mathbb{R}^{n\times k}$ and R^TAP is nonsingular, then the above minimum is achieved at the solution of the linear system

$$R^TAP\boldsymbol{x}_c = R^T\boldsymbol{b}. \tag{1.7}$$

We call R a *restriction matrix*, and we choose it to be whatever is useful.

One theoretically convenient choice is $R = A^{-1}P$. Then Equation (1.7) becomes

$$\boldsymbol{x}_c = P^TA^{-1}\boldsymbol{b} \tag{1.8}$$

and we find $\boldsymbol{x}_c$ immediately. This can be viewed as the orthogonal projection of $\boldsymbol{x}$ onto span(P).

In practice, it is usually infeasible to set $R = A^{-1}P$, as that would require already having a means of solving linear systems with A. Instead, we simply choose interpolation matrices that approximate $A^{-1}P$ well enough for our purposes. A common choice is $R = P$. Then $A_c = P^TAP$, which is symmetric and positive semi-definite. This then gives us

$$\boldsymbol{x} \approx PA_c^{-1}P^T\boldsymbol{b}. \tag{1.9}$$

Note the similarity between this equation and Equations (1.5) and (1.6): all of them involve projecting the problem onto a smaller matrix, solving a size-k system of equations, and then interpolating the matrix back to the original problem.

1.4.1 Use in Iterative Methods

In general, we do not know in advance an interpolation space span(P) containing the exact solution. Thus, a coarse-grid approximation introduces some error; that is, $PA_c^{-1}P^T\boldsymbol{b} - A^{-1}\boldsymbol{b}$ is nonzero. *Multigrid methods* are an example of error flattening that combines the above step with another method designed to reduce the new error introduced by the application of Equation (1.9).

As a concrete example of this approach, suppose that L is a graph Laplacian based on a 2D grid graph. Suppose further that we have a target $\boldsymbol{b}$ and a current guess $\boldsymbol{x}_t$. We wish to develop an iterative scheme to move $\boldsymbol{x}_t$ closer to the solution $\boldsymbol{x}^*$. The Jacobi method is the linear procedure

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \omega D^{-1}(\boldsymbol{b} - L\boldsymbol{x}_t) = \boldsymbol{x}_t + \omega D^{-1}\boldsymbol{r}_t,$$

where $D = \text{diag}(L)$ and ω is a positive scalar, typically around 2/3. When the underlying graph is a grid, the Jacobi method tends to shrink highly oscillatory components of the error vector, leaving the smooth error components largely untouched. For this reason, the Jacobi method is referred to as a *smoother*.

Because the smooth error components only change a little each iteration, solving a linear system with just the Jacobi preconditioner will take a long time, even if used inside an accelerator such as the conjugate gradient method [14]. Instead, we can pair the Jacobi smoother with another that addresses the smooth error components, switching back and forth between the two.

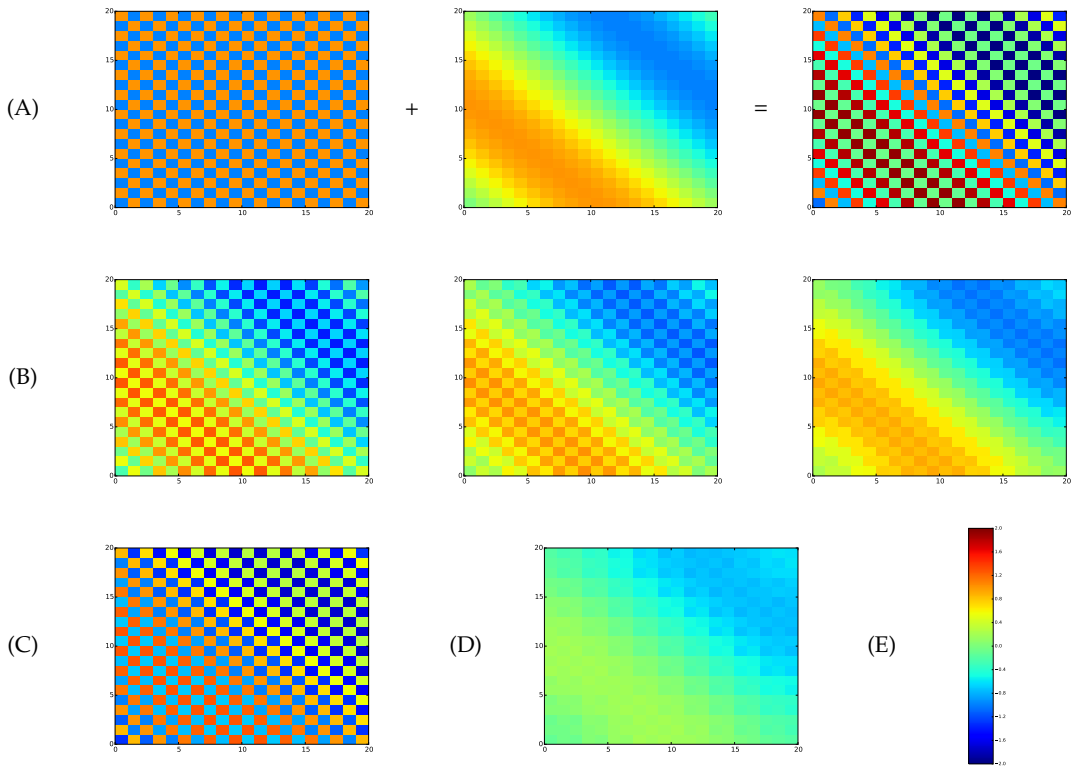


Figure 1.2: Smoothing and coarse-grid corrections on a 2D grid. (A) The sum of high-frequency and low-frequency errors on the 2D grid. (B) The error after few applications of the Jacobi method. Note the similarity to the low-frequency error. (C) The error if a coarse-grid correction is used instead. Note that the result is similar to the high-frequency error. (D) The error if a coarse-grid correction is applied after a few Jacobi iterations. (E) Color key.

In particular, suppose the node indices of the 2D grid graph are ordered such that every 4 indices represents a 2-by-2 square of nodes. Let

$$P = \begin{bmatrix} \mathbf{1}_4 & & & \\ & \mathbf{1}_4 & & \\ & & \ddots & \\ & & & \mathbf{1}_4 \end{bmatrix} \in \mathbb{R}^{n \times k}, \tag{1.10}$$

where $\mathbf{1}_4$ is the length-4 all ones vector. We will use this matrix to construct

a coarse-grid projection. The matrix $L_c = P^T LP$ is, in this case, also a graph Laplacian, and represents a smaller, *coarse grid* version of the original grid. The vector $P^T \boldsymbol{r}_t$ is the vector in which each element is the sum of the 8 underlying elements. The result is that L_c and $P^T \boldsymbol{r}_t$ are a "zoomed out" view of the residual and grid. So, smooth error modes that change over long distances are still well-represented, while high-frequency error modes disappear. This is fine, however, as the Jacobi iteration handles the highly-oscillatory error components that are not represented on the coarse grid.

So, the update

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + PL_c^{-1} P^T \boldsymbol{r} = (I - PL_c^{-1} L)\boldsymbol{x}_{t-1} + PL_c^{-1} P^T \boldsymbol{b}$$

is effective at shrinking the error components that the Jacobi iteration misses. As stated above, this projection is imperfect, and so it tends to reintroduce some error in the directions previously addressed by the Jacobi update. So, we iterate by switching back and forth between the Jacobi smoother, and the coarse-grid projection. In multigrid literature, the coarse solve is sometimes called a *coarse-grid correction*. An illustration of this two-step procedure on 2D grids is illustrated in Figure 1.2.

Although the Jacobi smoother with the coarse-grid correction is highly effective for linear systems on grids, this basic recipe can be adapted to other settings as well. In general, this approach is useful as long as the coarse-grid correction is paired with another complementary method. In Chapter 3, we use this approach to solve a system of equations that, while nonlinear, share many properties with linear systems of graph Laplacians. In Chapter 4, we again look at linear graph Laplacian systems, but we consider a much more general class of graphs besides grids. There we find that a different type of smoother also

pairs well with the coarse-grid projection.

CHAPTER 2

FLIER: PRACTICAL TOPOLOGY ERROR CORRECTION USING SPARSE PMUS

Colin Ponce and David Bindel

In this paper, we present a Fingerprint Linear Estimation Routine (FLiER) to identify topology errors in power networks using readings from sparsely-deployed phasor measurement units (PMUs). When a power line is removed from a network, or when a substation is reconfigured, the event leaves a unique "voltage fingerprint" of bus voltage changes that we can identify using only the portion of the network directly observed by the PMUs. The naive brute-force approach to identify a failed line from such voltage fingerprints, though simple and accurate, is slow. We derive an approximate algorithm based on a local linearization and a novel filtering approach that is faster and only slightly less accurate. We present experimental results using the IEEE 57-bus, IEEE 118-bus, and Polish 1999-2000 winter peak networks.

This paper has been submitted for publication to IEEE Transaction on Power Systems.

2.1 Introduction

2.1.1 Motivation

Topology error correction is an important component of a network monitoring and control system, and is a key part of the power grid state estimation pipeline, either as a pre-processing step or as an integrated part of a generalized state estimator. If a topology error processing module fails to correct an error, poor and even dangerous control actions may result [6, 15], as unexpected topology changes, such as those due to failed lines, may put stress on the remaining lines and destabilize the network. Thus, it is important to identify topology changes quickly in order to take appropriate control actions.

The power grid is roughly divided into transmission networks, which transfers bulk power at higher voltage, and distribution networks, which delivers low-voltage power to final customers. Currently, it is difficult to find topology changes in low-voltage distribution networks. Often, utilities lack the monitoring equipment to directly detect line failures, and are unaware of issues until customers call to report power losses. The radial nature of many distribution networks makes the task easier, but distributed generation will eliminate much of this benefit. Consequently, distribution network state estimation techniques such as [45, 52, 53] are particularly vulnerable to topology changes, as less information is available to correct for them. Algorithms for topology error identification allow utilities to quickly detect such problems, and allow for accurate state estimation even when the topology changes.

Finding topology changes is simpler in transmission networks, as substa-

tions and transmission lines have sensors that directly report failures (or switch open/closed status). However, if a sensor malfunctions, then finding the topology change is again difficult. This can happen due to normal equipment malfunctions, or because a cyber-attacker wishes to mislead network operators. Although failure to correctly identify a topology error is less common in a transmission network, the stakes are higher: state estimation based on incorrect topology assumptions can lead to incorrect estimates, causing operators to overlook system instability, and in the worst case, leading to avoidable blackouts. Thus, it is important to have more than one way to monitor network topology.

2.1.2 Prior work

Topology error detection and correction has evolved together with state estimation; for a good overview of the state of the art in state estimation at the turn of the century, including a discussion of the role of topology estimation, we refer to the review article [49]. Early work on topology error detection and correction goes back at least two decades prior [46]. Work in the late 1980s [18, 76] showed that topology errors are reflected in the shape of the residual in a standard state estimator, and this can be used to diagnose topology errors. In 1993, [19] took a related approach to diagnose errors through a correlation index based on the sensitivity of residuals to topology changes. When state estimates are based on a robust loss function, such as the ℓ^1 or least absolute variation (LAV) loss, the residual tends to be large for a sparse selection of outlier equations associated with topology errors; this observation was employed as early as 1982 in the context of substation topology validation [34], and was subsequently taken by others [2, 61].

A 1991 paper described an approach in which closed circuit breakers are modeled by explicit equations tying together variables and by an associated set of slack variables or Lagrange multipliers, which may be interpreted as pseudo-measurements [50]; the authors subsequently applied this approach to state estimation and topology error detection [48]. The sparse tableau formulation common in early circuit modeling also uses an extended set of multiplier variables, and in [17] these multipliers were used to diagnose topology errors. In [5], a generalized state estimation approach was introduced, also based on a breaker-level model, but using localized expanded bus-section models only when needed. Other authors combined breaker-level models together with a robust estimation procedure based on Huber's loss function in [47], and introduced extended systems with a minimal set of multipliers corresponding to the edges in trees associated with each connected set of bus sections [31, 21]. Normalized Lagrange multipliers have been interpreted geometrically as cosines of angles to a subspace associated with suspect information due to a topology error in [44]. More recent work solves mixed integer programs with modern solver methods to simultaneously estimate analogue variables and binary topology variables [15, 74].

In the late 2000s, transmission operators in the US began significant deployments of phasor measurement units (PMUs), sensors that directly measure voltage and current phasors at a bus many times per second. Given the network's admittance properties, one can compute the voltage phasors of neighboring buses as well [55]. While most PMUs today are in the transmission grids, distribution grids are likely to see PMUs in the future as well [60, 59]. Initial work has been undertaken to incorporate PMU information into state estimators [78, 77], as well as to diagnose topology change events such as single line failures [67]

and multiple-line failures [68, 80, 81]; there has even been preliminary work on identification of power system topologies from PMU data without a prior topology model [57].

PMU-based methods are not yet a replacement for existing SCADA-based state and topology estimators, particularly as current PMU deployments do not offer full observability of most of the grid. However, PMU-based approaches to state estimation, topology error detection, and situational awareness are a valuable supplement to existing methods. For example, while PMU-based methods can rapidly identify external transmission line outages [67], the system data exchange (SDX) model of the North American Electric Reliability Corporation (NERC) provides inter-area topology information only on an hourly basis [70]. Even within an operator's internal region, PMU-based methods may distinguish multiple topology change events that would occur between standard state estimates. Not only does the sequence of such events provide useful clues about causality, but the information about the collection of changes may be used as evidence in subsequent traditional topology error correction methods.

2.1.3 Our work

We present here an efficient method to identify topology changes in networks with a (possibly small) number of PMUs. We assume that a complete state estimate is obtained shortly before a topology change, e.g. through conventional SCADA measurements, and we use discrepancies between this state estimate and PMU measurements to identify failures. Note that while state estimation is not currently common in distribution networks, we can expect the advent of

smart grids to change this. Our method does not require complete observability from PMU data; it performs well even when there are few PMUs in the network, though having more PMUs does improve the accuracy. Though our approach is similar in spirit to [67], this paper makes three key novel contributions:

- Besides treating line failures, we show how our approach also applies to substation reconfigurations that result in bus splitting or merging. While this is not new for standard topology estimators, to our knowledge it is new for PMU-based methods.
- We describe a novel subspace-based filtering method capable of ruling out many candidate topology changes at relatively low cost.
- Rather than the DC approximation used in prior work on PMU-based line failure detection, we take advantage of existing state estimation procedures by linearizing the full AC power flow about a previously estimated state. This increases accuracy compared to the DC approximation.

2.2 Problem Formulation and Fingerprints

Let $y_{ik} = g_{ik} + \jmath b_{ik}$ denote the elements of the admittance matrix $Y \in \mathbb{C}^{n\times n}$ in a bus-branch network model; let P_ℓ and Q_ℓ denote the real and reactive power injections at bus ℓ; and let $v_\ell = |v_\ell| \exp(\jmath\theta_\ell)$ denote the voltage phasor at bus ℓ. These quantities are related by the power flow equations

$$H(v; Y) - s = 0 \tag{2.1}$$

where

$$\begin{bmatrix} H_\ell \\ H_{n+\ell} \end{bmatrix} = \sum_{h=1}^{n} |v_\ell||v_h| \begin{bmatrix} g_{\ell h} & b_{\ell h} \\ -b_{\ell h} & g_{\ell h} \end{bmatrix} \begin{bmatrix} \cos(\theta_{\ell h}) \\ \sin(\theta_{\ell h}) \end{bmatrix}, \tag{2.2}$$

with $\theta_{\ell h} = \theta_\ell - \theta_h$ and

$$s = \begin{bmatrix} P_1 & \cdots & P_n & Q_1 & \cdots & Q_n \end{bmatrix}^T. \tag{2.3}$$

We note that H is quadratic in v, but linear Y.

In a breaker-level model, we use a similar system in which variables are associated with bus sections, and H represents the power flows when all breakers are open. We then write the power flow equations as

$$\begin{aligned} H(v; Y) + C\lambda - s &= 0 \\ C^T v &= b, \end{aligned} \tag{2.4}$$

where the constraint equations $C^T v = b$ have the form

$$c_k^T v = (e_i - e_j)^T v = v_i - v_j = b_k = 0,$$

i.e. voltage variable j for a "slave" bus section is constrained to be the same as voltage variable i for a "master" bus section. In addition, we include constraints of the form

$$c_k^T v = e_i^T v = b_k$$

to assign a voltage magnitude at a PV bus or the phase angle at a slack bus. We could trivially eliminate these constraints, but keep them explicit for notational convenience.

Our goal is to use the power flow equations to diagnose topology changes such as single line failures or substation reconfigurations. We assume the network remains stable and the state shifts from one quasi-steady state to another.

As it does, the voltage vector shifts from v to $\hat{v} = v + \Delta v$. We assume m voltage phasor components, indicated by the rows of $E \in \{0,1\}^{m \times n}$, are directly observed by PMUs. Assuming the loads and generation remain roughly constant, we can predict what $E\Delta v$ should be for each possible contingency. That is, we can match the observed voltage changes $E\Delta v$ to a list of *voltage fingerprints* to identify simple topology changes.

It is possible that two or more contingencies have either the same or practically indistinguishable fingerprints. For example, one of two parallel lines with equal admittance may fail, or two lines that are distant from all PMUs but near each other may yield similar fingerprints. Often, even when a contingency is not identifiable, our method still produces valuable information. When multiple lines have the same effect on the network, our technique can be used to identify a small set of potential lines or breakers to inspect more closely.

2.3 Approximate Fingerprints

To compute the exact fingerprint for a contingency, we require a nonlinear power flow solve. In a large network with many possible contingencies, this computation becomes expensive. We approximate the changing voltage in each contingency by linearizing the AC power flow equations about the pre-contingency state. As in methods based on the DC approximation, we use the structure of changes to the linearized system to compute voltage change fingerprints for each contingency with a few linear solves. By using information about the current state, we observe better diagnostic accuracy with our AC linearization than with the DC approximation.

We consider three different types of contingencies: bus merging or bus splitting due to substation reconfiguration, and line failure. In each case, we assume the pre-contingency state is $x = (v, \lambda)$ satisfying (2.4). We denote the post-contingency state by primed variables $x' = (v', \lambda')$; we assume in general that the power injections s are the same before and after the contingency. The exact shift in state is $\Delta x' = x' - x$, and our approximate fingerprints are based from the approximation $\delta x' \approx \Delta x$ to the shift in state. The computation of $\delta x'$ for each contingency involves the pre-contingency Jacobian matrix

$$A = \begin{bmatrix} \frac{\partial H}{\partial v}(v; Y) & C \\ C^T & 0 \end{bmatrix}.$$

We assume a factorization of A is available, perhaps from a prior state estimate.

2.3.1 Bus Merging Fingerprints

In the case of two bus sections becoming electrically tied due to a breaker closing, we augment C by two additional constraints C' to tie together the voltage magnitudes and phase angles of the previously-separate bus sections. That is, the post-contingency state satisfies the augmented system

$$\begin{aligned} H(v'; Y) + C\lambda' + C'\gamma - s &= 0 \\ C^T v' &= b \\ C'^T v' &= 0. \end{aligned} \tag{2.5}$$

We linearize (2.5) about the original state x (with $\gamma = 0$); because the first two equations are satisfied at this state, we have the approximate system

$$\begin{bmatrix} A & U \\ U^T & 0 \end{bmatrix} \begin{bmatrix} \delta x' \\ \gamma \end{bmatrix} = - \begin{bmatrix} 0 \\ C'^T v \end{bmatrix}, \quad U = \begin{bmatrix} C' \\ 0 \end{bmatrix} \tag{2.6}$$

We then solve the system by block elimination to obtain

$$\gamma = (U^T A^{-1} U)^{-1} (C'^T v) \tag{2.7}$$

$$\delta x' = -A^{-1} U \gamma \tag{2.8}$$

The formulas (2.7)–(2.8) only require two significant linear solves (to evaluate $A^{-1}U$), some dot products, and a 2×2 solve.

2.3.2 Bus Splitting Fingerprints

When a bus splits after a breaker opens, the post-contingency state satisfies the augmented system

$$\begin{aligned} H(v'; Y) + C\lambda' - s &= 0 \\ C^T v' + F\gamma &= b \\ F^T \lambda' &= 0. \end{aligned} \tag{2.9}$$

The slack variables γ let the voltage phasor for a "breakaway" group of previously-slaved sections differ from a phasor at the former master section. The two columns of $F \in \{0,1\}^{n \times 2}$ indicate rows of C^T that constrain the breakaway voltage magnitudes and the phase angles, respectively. The third equation says no power flows across the open breaker.

We linearize (2.9) about the original state x (with $\gamma = 0$); because the first two equations are satisfied at this state, we have the approximate system

$$\begin{bmatrix} A & U \\ U^T & 0 \end{bmatrix} \begin{bmatrix} \delta x' \\ \gamma \end{bmatrix} = - \begin{bmatrix} 0 \\ F^T \lambda \end{bmatrix}, \quad U = \begin{bmatrix} 0 \\ F \end{bmatrix}. \tag{2.10}$$

The bordered systems (2.10) has the same form as (2.6); and, as before, block Gaussian elimination requires only two solves with A, some dot products, and a 2×2 system solve.

2.3.3 Line Failure Fingerprints

In principle, line failures can be handled in the same way as substation reconfigurations that lead to bus splitting: explicitly represent two nodes on a line that are normally connected (physically corresponding to two sides of a breaker) with a multiplier that forces them to be equal, and compute the fingerprint by an extended system that negates the effect of that multiplier. In practice, we may prefer to avoid the extra variables in this model. The following formulation requires no explicit extra variables in the base model, and can be used with either a breaker-level model or a bus-branch model with no breakers (i.e. C an empty matrix).

For line failures, the admittance changes to $Y' = Y + \Delta Y'$ where $\Delta Y'$ is a rank-one update. The post-contingency state satisfies the system

$$\begin{aligned} H(v'; Y') + C\lambda' - s &= 0 \\ C^T v' &= b, \end{aligned}$$

and linearization about x gives

$$\begin{bmatrix} \frac{\partial H}{\partial v}(v; Y') & C \\ C^T & 0 \end{bmatrix} \begin{bmatrix} \delta v' \\ \delta \lambda' \end{bmatrix} = - \begin{bmatrix} H(v; \Delta Y') \\ 0 \end{bmatrix} \tag{2.11}$$

where $H(v; \Delta Y') = H(v; Y') - H(v; Y)$. As we show momentarily,

$$\frac{\partial H}{\partial v}(v; Y') - \frac{\partial H}{\partial v}(v; Y) = \frac{\partial H}{\partial v}(v; \Delta Y') = U^0 (V^0)^T.$$

where U^0 and V^0 each have three columns. That is, the matrix in the system (2.11) is a rank-three update to A. We can solve such a system by the Sherman-Morrison-Woodbury update formula, also widely known as the Inverse Matrix Modification Lemma [4, 32]. We use the equivalent extended sys-

tem

$$\begin{bmatrix} A & U \\ V^T & -I \end{bmatrix} \begin{bmatrix} \delta x' \\ \gamma \end{bmatrix} = - \begin{bmatrix} r \\ 0 \end{bmatrix}, \tag{2.12}$$

where

$$U = \begin{bmatrix} U^0 \\ 0 \end{bmatrix}, \quad V = \begin{bmatrix} V^0 \\ 0 \end{bmatrix}, \quad r = \begin{bmatrix} H(v; \Delta Y') \\ 0 \end{bmatrix}.$$

We again solve by block elimination:

$$\gamma = (I + V^T A^{-1} U)^{-1} (V^T r) \tag{2.13}$$

$$\delta x' = -A^{-1}(r + U\gamma). \tag{2.14}$$

The work to evaluate (2.13)–(2.14) is three linear solves (for $A^{-1}U$), some dot products, and a small 3×3 solve. We will show momentarily how to avoid the solve involving r.

We now show that the Jacobian matrix changes by a rank-3 update. For a failed line between nodes i and k, the vector $H(v, \Delta Y')$ has only four nonzero entries:

$$\begin{aligned} \check{P}_i \equiv H_i &= \check{P}_{ik} + g'_{ii}|v_i|^2 \\ \check{Q}_i \equiv H_{i+n} &= \check{Q}_{ik} - b'_{ii}|v_i|^2 \\ \check{P}_k \equiv H_k &= \check{P}_{ki} + g'_{kk}|v_k|^2 \\ \check{Q}_k \equiv H_{k+n} &= \check{Q}_{ki} - b'_{kk}|v_k|^2, \end{aligned}$$

where

$$\begin{bmatrix} \check{P}_{ik} \\ \check{Q}_{ik} \end{bmatrix} \equiv |v_i||v_k| \begin{bmatrix} g'_{ik} & b'_{ik} \\ -b'_{ik} & g'_{ik} \end{bmatrix} \begin{bmatrix} \cos(\theta_{ik}) \\ \sin(\theta_{ik}) \end{bmatrix},$$

and $\check{P}_{ki}, \check{Q}_{ki}$ are defined similarly. Let

$$D_{ik} \equiv \frac{\partial(\check{P}_i, \check{P}_k, \check{Q}_i, \check{Q}_k)}{\partial(\theta_i, \theta_k, |v|_i, |v|_k)} \in \mathbb{R}^{4\times 4};$$

by the chain rule, we can write $D_{ik} = U_{ik} V_{ik}^T$ where

$$U_{ik} \equiv \frac{\partial(\check{P}_i, \check{P}_k, \check{Q}_i, \check{Q}_k)}{\partial(\theta_{ik}, \log|v|_i, \log|v|_k)} \in \mathbb{R}^{4\times 3}$$
$$V_{ik}^T \equiv \frac{\partial(\theta_{ik}, \log|v|_i, \log|v|_k)}{\partial(\theta_i, \theta_k, |v|_i, |v|_k)} \in \mathbb{R}^{3\times 4}.$$

More concretely, we have

$$U_{ik} = \begin{bmatrix} -\check{Q}_i - b'_{ii}|v_i|^2 & \check{P}_i + g'_{ii}|v_i|^2 & \check{P}_i - g'_{ii}|v_i|^2 \\ \check{Q}_k + b'_{kk}|v_k|^2 & \check{P}_k - g'_{kk}|v_k|^2 & \check{P}_k + g'_{kk}|v_k|^2 \\ \check{P}_i - g'_{ii}|v_i|^2 & \check{Q}_i - b'_{ii}|v_i|^2 & \check{Q}_i + b'_{ii}|v_i|^2 \\ -\check{P}_k + g'_{kk}|v_k|^2 & \check{Q}_k + b'_{kk}|v_k|^2 & \check{Q}_k - b'_{kk}|v_k|^2 \end{bmatrix}$$
$$V_{ik}^T = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & |v_i|^{-1} & 0 \\ 0 & 0 & 0 & |v_k|^{-1} \end{bmatrix}.$$

Because $H(v, \Delta Y')$ does not depend on any voltage phasors other than those at nodes i and j, we may write

$$\frac{\partial H(v; \Delta Y')}{\partial v} = E_{ik} D_{ik} E_{ik}^T = U^0 (V^0)^T \tag{2.15}$$

where

$$E_{ik} = \begin{bmatrix} e_i & e_k & & \\ & & e_i & e_k \end{bmatrix} \in \mathbb{R}^{2n\times 4}. \tag{2.16}$$

and

$$U^0 = E_{ik} U_{ik}, \quad V^0 = E_{ik} V_{ik}. \tag{2.17}$$

Moreover, we note that

$$H(v, \Delta Y') = E_{ik} \begin{bmatrix} \check{P}_i \\ \check{P}_k \\ \check{Q}_i \\ \check{Q}_k \end{bmatrix} = U^0 z, \quad z = \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \end{bmatrix},$$

so that we may rewrite (2.14) as

$$\delta x' = -A^{-1}U(z + \gamma). \tag{2.18}$$

2.4 Filtering

In Section 2.3 we discussed how to approximate voltage shifts $\delta v'$ associated with several types of contingencies. This approach to predicting voltage changes costs less than a nonlinear power flow solve, but may still be costly for a large network with many contingencies to check. In the current section we show how to rule out contingencies without any solves by computing a cheap lower bound on the discrepancy between the observed voltage changes and the predicted voltage changes under the contingencies.

For each contingency, we define the *fingerprint score*

$$t = \|E\Delta v - E\delta v'\| \tag{2.19}$$

where Δv is the observed voltage shift and $\delta v'$ is the voltage shift predicted for the contingency. For the contingencies we have described, $E\delta v'$ has the form

$$E\delta v' = \bar{E}A^{-1}U\gamma \tag{2.20}$$

where $\bar{E} = \begin{bmatrix} E & 0 \end{bmatrix}$ simply ignores the multiplier variables λ, and γ is some short vector of slack variables. The expression $\bar{E}A^{-1}$ does not depend on the contingency, and can be pre-computed at the cost of m linear solves (one per observed phasor component). After this computation, the main cost in evaluating (2.20) is the computation of γ, which involves a contingency-dependent linear system with A as an intermediate step. However, we do not need γ for the *filter score*

$$\tau = \min_{\mu} \|E\Delta v - \bar{E}A^{-1}U\mu\| \leq t. \tag{2.21}$$

In the Euclidean norm, τ is simply the size of the residual in a least squares fit of $E\Delta v$ to the columns of $\bar{E}A^{-1}U$, which can be computed quickly due to the sparsity of U. If U is the augmentation matrix associated with contingency i, we refer to $\bar{E}A^{-1}U$ as its *filtering subspace*.

Filter score computations are cheap; and if the filter score τ_i for contingency i exceeds the fingerprint score t_k for contingency k, then we know

$$t_k < \tau_i \leq t_i,$$

without ever computing t_i. Exploiting this fact leads to the FLiER method (Algorithm 1).

Algorithm 1: FLiER

Compute and store $\bar{E}A^{-1}$.

For each contingency i, compute τ_i via (2.21).

Order the contingencies in ascending order by τ.

for $\ell = 2, 3, \ldots$ **do**

  Compute fingerprint score t_ℓ

  Break if $t_\ell < \tau_{\ell+1}$

end for

Return contingencies with computed t_ℓ

Filter score computations are embarrassingly parallel and can be spread across processors. Nonetheless, for huge networks with many PMUs and many contingencies, the filter computations might be deemed too expensive for very rapid diagnosis (e.g. in less than a second). However, the concept of a filter

Example Python code of this algorithm can be found at `https://github.com/cponce512/FLiER_Test_Suite`

subspace can be adapted to these cases. First, one can define a *coarse* filtering subspaces that is the sum of the filtering subspaces for a set of contingencies. For example, one might define a coarse filtering subspace associated with all possible breaker reconfigurations inside a substation. The coarse subspace filter score provides a lower bound on the filter scores (and hence the fingerprint scores) for all contingencies in the set. Hence, it may not even be necessary to compute individual filter scores for all contingencies considered. Second, one can work with a *projected* filtering subspace $W^T(EA^{-1}U)$ where W is a matrix with orthonormal columns. The distance from a projected measurement vector to the projected filtering subspace again gives a lower bound on the full filter score. In addition to reducing the cost of filter score computations, projections can also be used to eliminate faulty or missing PMU measurements from consideration.

2.5 Experiments

Our standard experimental setup is as follows. For each possible topology change, we compute and pass to FLiER both the full pre-contingency state and the subset of the post-contingency state that would be observed by the PMUs. We test both with no noise and with independent random Gaussian noise with standard deviation $1.7 \cdot 10^{-3}$ (≈ 0.1 degrees for phase angles) added to both the initial state estimate and the PMU readings. In [67], 0.1 degrees of Gaussian random noise was applied to phase angles, then smoothed by passing a simulated time-domain signal through a low pass filter; we apply the noise without filtering, so the effect is more drastic.

One of the possibilities FLiER checks is that there has been no change; in

this case, we use the norm of the fingerprint as both the fingerprint score and the filter score. By including this possibility among those checked, FLiER acts simultaneously as a method for topology change detection and identification.

We run tests on the IEEE 57 bus and 118 bus networks, with three different PMU arrangements on each:

- **Single**: Only one PMU is placed in the network, at a low-degree node (bus 35 in the 57-bus network and 65 in the 118-bus network). This represents a near-worst-case deployment for our method.
- **Sparse**: A few PMUs are placed about the network (on buses 4, 13, and 34 in the 57-bus network and on buses 5, 17, 37, 66, 80, and 100 in the 118-bus network). We consider this a realistic scenario in which sparsely-deployed PMUs do not offer full network observability.
- **All**: PMUs are placed on all buses. Any error is due purely to the linear approximation.

We did not test changes that cause convergence failure in our power flow solver. We assume such contingencies result in collapse without some control action.

2.5.1 Accuracy

Line Failures

Figure 2.1 shows the accuracy of FLiER in identifying line failures in the IEEE 57-bus test network. For each PMU deployment, we show the cumulative distribution function of ranks, i.e. the ranks of each simulated contingency in the

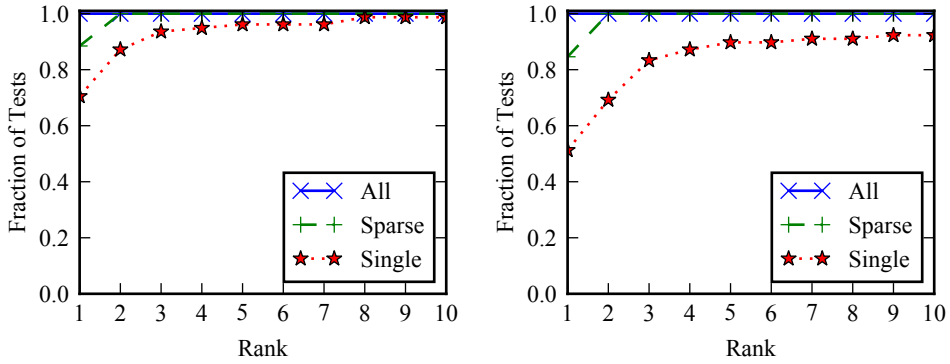


Figure 2.1: Cumulative distribution function showing the fraction of line failures where FLiER assigned the correct line at most a given rank (up to 10). Top: Noise-free case. Bottom: Entries with Gaussian noise with $\sigma = 0.0017$.

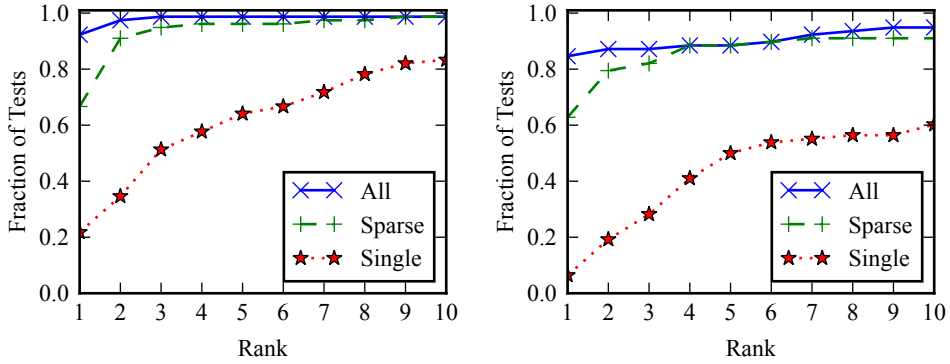


Figure 2.2: CDF of line failures where the DC approximation of [67] assigned the correct line at most a given rank (up to 10). Top: Noise-free case. Bottom: Entries with Gaussian noise with $\sigma = 0.0017$.

ordered list produced by FLiER. We show further results in Table 2.1. With PMUs everywhere, the correct answer was chosen in all 78 of 78 cases, even in the presence of noise. The case with three PMUs is also quite robust to noise. In the test with a single unfavorably-placed PMU, FLiER typically ranks the correct line among the top three in the absence of noise; with noise, the accuracy degrades, though not completely.

PMUs	Single	Sparse	All
FLiER	55(73)	68(77)	78(78)
FLiER+noise	40(65)	66(78)	78(78)
DC Approx	17(40)	52(74)	72(77)
DC Approx+noise	5(22)	49(64)	66(68)

Table 2.1: IEEE 57-bus network accuracy comparison for 78 line failure contingencies. We report counts of line failures correctly identified and those scored in the top three (in parentheses).

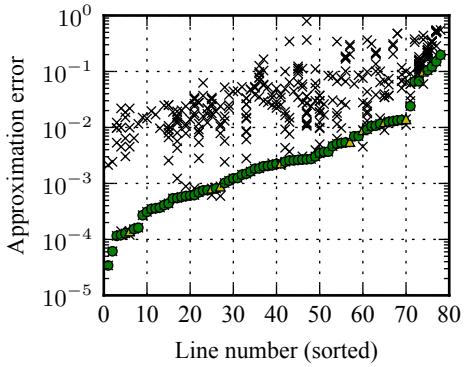


Figure 2.3: Test of our algorithm on the IEEE 57-bus network with the sparse PMU deployment. Each column represents one test. Black crosses are fingerprint scores for incorrect lines. Green dots and yellow triangles indicate the scores of the correct line in the case of correct diagnosis or diagnosis in the top three, respectively.

In Figure 2.2, we repeat the experiment of Figure 2.1, but with the DC approximation used in [67] rather than the AC linearization used in FLiER. We also present comparisons in Table 2.1. With PMUs everywhere, there is little difference in accuracy. With fewer PMUs, FLiER is more accurate. In the sparse case, the DC approximation without noise behaves similarly to FLiER with noise, while in the single PMU deployment the DC results without noise are much worse than those from FLiER even with noise.

Figure 2.3 shows the raw scores computed by FLiER with three PMUs. In

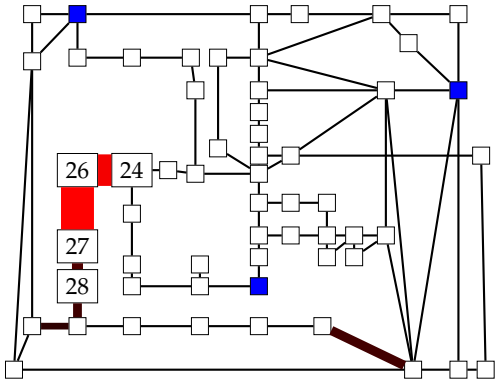


Figure 2.4: Line (24, 26) is the line removed in this test. Lines are colored and thickened according to $\sqrt{t_{ik}^{-1}}$. Line (26, 27) was chosen by the algorithm.

this plot, each column represents the fingerprint scores computed for one line failure scenario. The black crosses represent the scores of lines that get past the filter, while the green circles and yellow triangles represent the scores for the correct answer. If there is a green circle, then our algorithm correctly identified the actual line that failed. If there is a yellow triangle, the correct line was not chosen but was among the top three lines selected by the algorithm.

In Figure 2.4, we show one case that FLiER misidentifies. PMUs are deployed on buses marked with blue squares, and lines are colored and thickened according to the FLiER score. The best-scoring line is adjacent to the line that failed.

Substation Reconfigurations

Next, we show the accuracy of FLiER as it applies to substation reconfigurations. For these tests, we suppose that every bus in the IEEE 57-bus test network is a ring substation with each bus section on the ring possessing either load, gen-

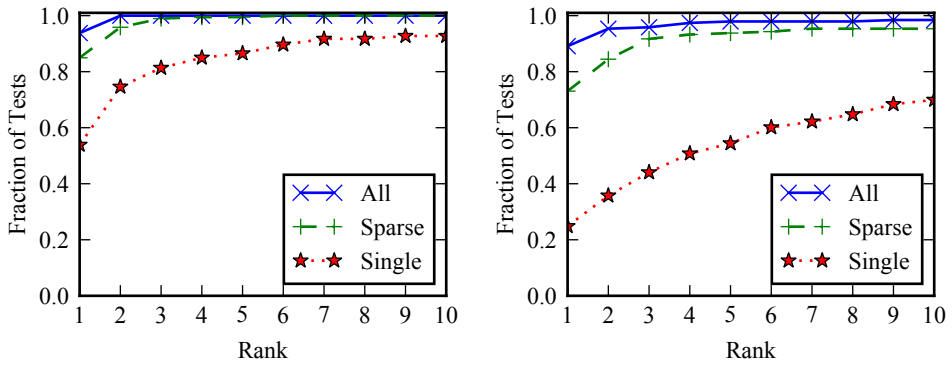


Figure 2.5: Rank CDF for substation reconfigurations without noise (top) and with noise (bottom)

eration, or a branch. We then suppose a substation splits when two of its circuit breakers open. We do not consider cases that isolate a node with a nonzero power injection. Line failures are a subset of this scenario: if the breakers on either side of a section with a branch open, that section becomes a zero-injection leaf bus, which disappears in the quasi-static setting.

Figure 2.5 shows the accuracy of FLiER on substation reconfigurations with and without noise. With three PMUs and no noise, FLiER is right in 164 of 193 possibilities, and ranks the correct answer among the top three scores in 160 cases. With PMUs everywhere, FLiER is right 181 times, but gets the answer in the top three every single time. With few PMUs, FLiER is more susceptible to noise when diagnosing substation reconfigurations. This is expected, as there are significantly more possibilities to choose from in this case. Also, FLiER sometimes filters out the correct answer in the presence of noise. One possible remedy for this would be to be more lenient with filtering, only throwing a possibility away if τ_ℓ is greater than the kth smallest t_ℓ, for example.

Finally, we demonstrate the effectiveness of using FLiER for substation reconfigurations on a large-scale network by running FLiER on the 400, 220, and

	Contingency	Substation of contingency
Correct %	75.2	85.4
Top 3 %	95.4	96.5

Table 2.2: Accuracy of FLiER with 100 randomly-placed PMUs on the Polish network. Results are out of 6283 tests.

100 kV subset of the Polish network during peak conditions of the 1999-2000 winter, taken from [82]. This is a larger network with 2,383 buses. We placed 100 PMUs randomly around the network, and tested every substation reconfiguration contingency. We summarize the results in Table 2.2. We could likely further improve the accuracy with a thoughtful deployment of PMUs.

2.5.2 Filter Effectiveness and Speed

The cost of FLiER depends strongly on the effectiveness of the filtering procedure. In Figure 2.6, we show how often the filter saves us from computing fingerprint scores in experiments on the IEEE 57-bus and 118-bus networks when checking for line failures. For each PMU deployment, we show the cumulative distribution function of the fraction of lines for which fingerprint scores need not be computed for each line failure. The filter performs well even for the sparse PMU deployments; we show a typical case in Figure 2.7.

Finally, we demonstrate the importance of the filter by running FLiER on the large Polish network [82] with 100 randomly placed PMUs. Table 2.3 shows FLiER run times with and without the filter on ten randomly selected branches. The code is unoptimized Python, so these timings do not indicate of how fast FLiER would run in a performance setting. However, they give a sense of the

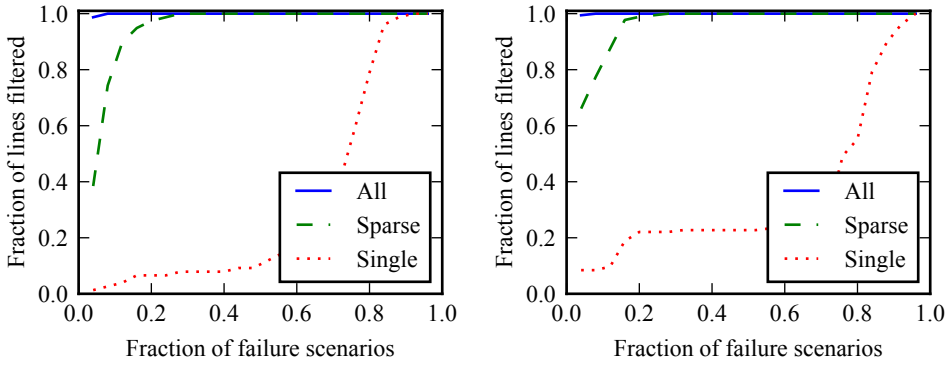


Figure 2.6: Cumulative distribution function of fraction of lines for which t_{ik} need not be computed when a line in the IEEE 57-bus (top) or 118-bus (bottom) network fails uniformly at random.

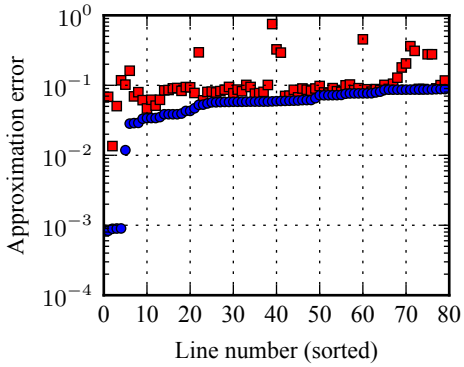


Figure 2.7: Example of effective filtering in Algorithm 1. Each column represents a line checked. Blue dots are the lower bounds τ_{ik}, while red squares are true scores t_{ik}. Columns are sorted by τ_{ik}. In this case, t_{ik} only needs to be computed for eight lines.

speedup one expects from filtering.

Note also that FLiER correctly identified the failed branch in 9 of 10 cases. In the one case in which it failed, on branch (2346, 2341), t_ℓ for the correct answer was $6.25 \cdot 10^{-5}$; this suggests the failure had a negligible impact on the network.

We also performed this timing test for a randomly selected set of substation reconfigurations, shown in Table 2.4. Again, the results give a sense of the

Line	FLiER (s)	Solution rank	# t's computed	FLiER n.f. (s)
(1502, 917)	0.27	1	2	14.14
(1502, 1482)	0.27	1	2	15.30
(557, 556)	0.27	1	4	14.75
(2346, 2341)	2.95	14	502	14.40
(909, 1155)	0.26	1	2	14.32
(644, 629)	0.29	1	7	14.59
(591, 737)	0.29	1	6	17.27
(559, 542)	0.32	1	13	16.59
(378, 336)	0.28	1	6	16.16
(101, 94)	0.26	1	2	15.29

Table 2.3: FLiER run times for ten line failures with and without filtering. About 3000 contingencies are considered.

Bus / Split nodes	FLiER (s)	Sol. rank / Sol. bus rank	# t's computed	FLiER n.f. (s)
86/1,2,3	4.54	1/1	2	823.0
176/1,2	4.70	3/3	4	836.9
539/7	8.11	1/1	38	829.8
702/4,5	4.66	1/1	4	820.1
754/2,3,4,5	4.97	1/1	7	829.8
994/2,3,4,5	4.86	1/1	6	835.8
1131/2	5.22	1/1	9	850.3
1513/4,5,6	6.65	4/1	23	862.3
1663/1,2,3,4	7.83	1/1	35	928.1
2164/5	4.56	1/1	3	875.3

Table 2.4: FLiER run times for ten substation reconfigurations with and without filtering. Nearly 7000 contingencies are considered.

speedup one expects from filtering in the substation reconfiguration case.

2.6 Conclusion and Future Work

We have presented FLiER, a new algorithm to identify topology errors involving failed lines and substation reconfigurations using a sparse deployment of

PMUs. Our method uses a linearization of the power flow equations together with a novel subspace-based filtering approach to provide fast diagnosis. Unlike prior approaches based on DC approximation, our approach takes advantage of a state estimate obtained shortly before the topology changes, assuming that the network specifications remain unchanged or change in a known way as a result of the failure. Our method can be used in both distribution and transmission networks, and is compatible with different levels of network model detail.

Several extensions remain open for future work. We hope to model noise sensitivity of our computations, so that we can provide approximate confidence intervals for fingerprint and filter scores; we also believe it possible to diagnose when the linear approximation will lead to incorrect diagnosis, and do more computation to deal just with those cases. In addition, we plan to extend our approach to other events, such as single-phase line failures or changes in line parameters due to overloading.

CHAPTER 3

NONLINEAR MULTIGRID FOR THE POWER FLOW EQUATIONS

Colin Ponce, Panayot Vassilevski, and David Bindel

Multigrid methods are extremely successful at solving certain classes of linear systems of equations. Today, much multigrid research focuses on extending the classes of problems to which one may apply these methods. This research can take two forms: extending the set of linear systems for which one may use multigrid, and finding classes of nonlinear problems for which one can use multigrid concepts. In this paper, we do the latter.

The power flow problem is a standard problem in power engineering. It is a complex quadratic system of equations that today is typically solved via Newton's method. However, as problem sizes grow larger, the Newton method requirement to repeatedly solve large linear systems is becoming a computational bottleneck. Here we develop an algebraic multigrid approach to address this problem. Our method produces a hierarchy of coarse power flow problems, and so can be used in conjunction with Newton's method at the coarse levels. We demonstrate the applicability of both Gauss-Seidel and Jacobi smoothers. Experiments demonstrate the highly scalable nature of our approach.

This paper has not yet been submitted for publication.

3.1 Introduction

Multigrid methods have seen great success as a powerful tool for the solution of linear systems of equations that arise from elliptic partial differential equations [11]. Linear multigrid methods have also been applied to linear systems that do not arise from partial differential equations, and often do not even have an underlying geometry [14]. This field of algebraic multigrid is still an active area of research [24, 54, 13].

The nonlinear Full Approximation Scheme (FAS) is a well-known multigrid framework for solving nonlinear partial differential equations [14]. Nonlinear multigrid methods have also been used to solve symmetric eigenvalue problems [38]. In this paper, we extend algebraic nonlinear multigrid methods to problems with nonlinear diagonal scaling of the equations in the system. In particular, we present a multigrid method for solving the *power flow problem*, also known as the load flow problem.

The power flow problem is standard in the field of power engineering. The power flow equations are a complex, quadratic system of equations that relate the voltages at each node in a power network to the amount of power entering or leaving at each node [30]. The simulation of a power network in "steady state" requires the solution of the power flow equations, while time-domain simulation requires solving the power flow equations at each time step in a differential-algebraic problem [65]. Power flow is an inherently nonlinear problem, and although power networks are physical networks, modeling data often does not include physical locations. Thus, it is typically an algebraic problem rather than a geometric one.

The earliest methods for solving the power flow problem were nonlinear Gauss-Seidel methods known as *Y-Matrix methods*, developed in the 1950's [64]. These methods converge slowly but were suitable for early computers with limited memory. They were surpassed in the 1960's by Newton-Raphson methods with the development of efficient sparse Gaussian elimination techniques that made Jacobian factorization practical [69].

Today Newton-Raphson is still the workhorse method for the power flow problem [30], with further developments typically focusing on special cases or on accelerating the Jacobian linear solve. The *Fast-Decoupled Load Flow* is a quasi-Newton method that makes assumptions about typical operating conditions to decouple the Jacobian into two smaller matrices [66]. The *DC Load Flow* method makes further assumptions by completely neglecting voltage magnitude changes to reduce the problem to a single linear solve [30]. The development of continuation methods enabled power flow computation near points of system instability [3].

Power engineers today typically model systems with up to about 100,000 nodes, but model sizes are expected to increase dramatically in coming years as operators begin to run larger and more detailed simulations [56, 33]. While Newton's method has excellent convergence properties, repeatedly factoring Jacobian matrices becomes a bottleneck at large problem sizes. As a result, much recent work has focused on the efficiency of the Jacobian solve through developments in sparse direct solvers [37, 42, 62] and Krylov solvers [36, 33, 79]. In [33], the authors briefly suggest algebraic multigrid as a preconditioner in a Newton-Krylov iteration.

In this paper, we do not use algebraic multigrid to solve a Jacobian linear

system, but present a nonlinear multigrid framework for the power flow problem itself. The result is a hierarchy of power flow problems of different sizes. We can use the full hierarchy to produce a completely Jacobian-free solver, or we can directly solve a power flow problem using Newton's method once it has reached an acceptably small size. The latter approach allows us to leverage the desirable convergence properties of Newton's method without needing to solve large, expensive linear systems.

At each level of the multigrid hierarchy, we employ a nonlinear iterative method. We discuss both nonlinear Gauss-Seidel and nonlinear Jacobi iterations. While the Gauss-Seidel method typically converges faster, the Jacobi iteration is highly parallelizable. To the author's knowledge, this is also the first successful use of a Jacobi iteration in solving the power flow problem.

The rest of this paper is organized as follows. In Section 3.2, we introduce the power flow equations and discuss current solution methods. In Section 3.3, we discuss some preliminary concepts. In Section 3.4, we introduce the nonlinear Gauss-Seidel iteration and its extension to group-synchronous updates. In Section 3.5, we show how to convert group-synchronous updates into a hierarchy of nested power flow problems. In Section 3.6, we develop convergence results for out method. In Section 3.7, we present numerical experiments, and in Section 4.8, we conclude.

3.2 The Power Flow Equations

One can represent a power network as a graph; power enters or leaves a network at nodes and flows between nodes along edges. These edges are transmis-

sion lines and are weighted by their electrical admittance. When simulating a power network that is in "steady state," power engineers typically make a set of simplifying assumptions.

1. *Complex numbers for voltages and currents.* Power grids use alternating current, in which all voltages and currents in a network oscillate at the same frequency (e.g. 60 Hz in the United States). We may represent these sine waves as complex phasors.
2. *Dimensionless units.* All quantities in a power network model are normalized, resulting in a unitless representation of electrical voltage, current, and power. The result is that all nodes have voltage magnitudes near 1.0. Voltage phase angles vary more than magnitudes, but still remain near 0.0.
3. *Constant electrical impedance of lines.* The impedance of transmission lines vary according to a number of factors, including current flow and ambient temperature. However, for steady state modeling, impedances are typically assumed to be constant.

Under these assumptions, we can model power networks by representing nodal voltages, transmission line admittances, and network power injections as complex numbers.

To set up the model equations, we first define the *admittance matrix* $Y \in \mathbb{C}^{n \times n}$. The off-diagonal element Y_{ij} is the negative of electrical admittance of the transmission line connecting nodes i and j, or zero if there is no such line. The diagonal elements Y_{ii} are the sum of all connected electrical admittances, as well as a small additional *shunt admittance* term Y_{is} [30]. It is nearly symmetric, except on

lines with transformers, which introduce asymmetries in phase angle but not magnitude. The result is that Y is nearly a complex graph Laplacian.

Let $\boldsymbol{v} \in \mathbb{C}^n$ denote a vector of complex voltages at each node and let $\boldsymbol{s} \in \mathbb{C}^n$ denote a vector of complex power injections at each node. In addition let $D_{\boldsymbol{v}} = \mathrm{diag}(\boldsymbol{v})$ and let $\bar{\boldsymbol{x}}$ denote the complex conjugate of a vector $\boldsymbol{x}$. Then, under the assumptions stated above, we have the relation

$$\boldsymbol{i} = Y\boldsymbol{v} \tag{3.1}$$

$$D_{\boldsymbol{v}}\bar{\boldsymbol{i}} = \boldsymbol{s}, \tag{3.2}$$

where $\boldsymbol{i}$ is the complex vector of nodal current injections. We write this relation in one line as $D_{\boldsymbol{v}}\overline{Y\boldsymbol{v}} = \boldsymbol{s}$. In the *power flow problem*, one is given $\boldsymbol{s}$ and Y and solves for $\boldsymbol{v}$.

There are three types of nodes in a power flow model:

1. *PQ Nodes.* In a PQ node i, the power injection $s_i = P_i + \imath Q_i$ is known a priori, and we wish to solve for the unknown voltage v_i. Approximately 90% of the nodes in a network are PQ nodes [39].
2. *PV Nodes.* In a PV node i, the real component of the power injection, $\mathrm{Re}\, s_i = P_i$ is known, as well as the voltage magnitude, which we denote $|v_i^{sp}|$. Here we wish to solve for the unknown voltage phase angle $\theta_i = \arg(v_i)$. PV nodes represent generators and usually make up approximately 10% of the nodes in a network [39].
3. *The Slack Node.* There is always one slack node. The complex voltage v_i is known a priori. Without a slack node, the power flow equations would be invariant to scalar complex rotations of $\boldsymbol{v}$ and consequently underdetermined.

Note that the power injection of the slack node and the imaginary component of the power injections of the PV nodes (that is, the reactive power injection) are also unknown. However, they are simple to compute once the nodal voltages are found.

3.3 The DC Approximation and Linear Multigrid

In this section we discuss the DC Load Flow approximation to the power flow equations and its connection to multigrid methods. We will use this discussion to motivate the algorithmic developments in Sections 3.4 and 3.5.

3.3.1 The DC Load Flow Approximation

The DC Load Flow approximation (see, e.g. [30]) is derived from a set of observations about typical operating conditions of power grids. These observations allow us to make reasonable simplifying approximations to the power flow equations that convert Equation (3.1) into a linear system of equations.

We make the following simplifying observations:

1. Transmission lines have significantly larger reactance than resistance. Therefore, $Y \approx \imath \operatorname{Im}(Y)$.
2. Typically, neighboring buses have voltage phase angles that are close to each other. This allows us to make the simplifying approximation $e^{\imath(\theta_k - \theta_i)} \approx 1 + \imath(\theta_k - \theta_j)$.
3. Typically, $0.95 < |v_k| < 1.05$. Therefore, we approximate $|v_k| \approx 1.0$.

The result of these simplifications is that the DC Load Flow approximation is only concerned with the real part of the power injections and the nodal voltage phase angles θ_k. A calculation then reduces Equation (3.1) to the linear system

$$\mathrm{Re}(\boldsymbol{s}) = B\boldsymbol{\theta},$$

where $\boldsymbol{\theta}$ is the vector of node voltage phase angles and

$$B_{ij} = \mathrm{Im}(Y_{ij})$$
$$B_{ii} = -\sum_{j \neq i} B_{ij}.$$

Note that the DC Load Flow also ignores any shunt admittance.

3.3.2 Multigrid for Graph Laplacian Systems

Multigrid methods are an effective means of solving large graph Laplacian-based linear systems when the graph is approximately grid-structured. It is an iterative method and does not require one to factor the matrix [14].

Suppose we want to solve

$$L\boldsymbol{x} = \boldsymbol{s},$$

where L is a graph Laplacian. Suppose one has a guess $\boldsymbol{x}_t$. The error here is $\boldsymbol{\epsilon}_t = \boldsymbol{x}_t - \boldsymbol{x}$. A multigrid method consists of two parts: a *smoother* and a *coarse-grid correction*. The smoother is a fast iterative procedure that does not necessarily shrink $\boldsymbol{\epsilon}_t$ by very much, but eliminates the high frequency error modes. The resulting smoothed $\boldsymbol{\epsilon}_t$ consists mostly of low-frequency error modes.

A smoother can be any of a number of algorithms, but a common option is the Gauss-Seidel method. It cycles through all variables, and for each variable i,

solves for x_i using the ith equation. That is, it finds x_i to satisfy the equation

$$0 = s_i - \sum_{j=1}^{n} L_{ij} x_j. \tag{3.3}$$

Once the error is smoothed with one or more smoother iterations, the next step is to shrink the remaining low frequency error. One does this by exploiting the fact that nearby nodes have similar errors. In particular, let P be the $n \times m$ matrix

$$P = \begin{bmatrix} \mathbf{1} & & \\ & \mathbf{1} & \\ & & \ddots \\ & & & \mathbf{1} \end{bmatrix}, \tag{3.4}$$

where each $\mathbf{1}$ is a sub-vector of all ones (typically of size 4 on a 2D grid). Then the assertion that nearby nodes have similar errors is the same as

$$\boldsymbol{\epsilon}_t \approx P\boldsymbol{x}_c$$

for some $\boldsymbol{x}_c \in \mathbb{R}^m$.

This suggests that

$$\begin{aligned} L(\boldsymbol{x}_t + P\boldsymbol{x}_c) &\approx \boldsymbol{s} \\ \implies LP\boldsymbol{x}_c &= \boldsymbol{s} - L\boldsymbol{x}_t. \end{aligned} \tag{3.5}$$

This is an overdetermined system, so one approximately solves it by applying a Galerkin projection; that is, one premultiplies the equation by a matrix to project the target space onto a size-m subspace. While many matrices are possible for this step, a common choice is P^T:

$$(P^T LP)\boldsymbol{x}_c = P^T(\boldsymbol{s} - L\boldsymbol{x}_t). \tag{3.6}$$

This is now a smaller, size-m system. One either completely or partially solves this *coarse system*, and then passes the result back to the original problem with the update

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + P\boldsymbol{x}_c.$$

This step is called the *coarse-grid correction*.

The matrix P is often chosen as the Galerkin projection matrix for two primary reasons. First, it allows one to characterize the smoother update and the coarse-grid correction using a *variational characterization*, in which the function

$$\|L\boldsymbol{x}_t - \boldsymbol{s}\|_{L^{-1}}^2$$

is minimized by iteratively choosing updates to $\boldsymbol{x}_t$ that lie within a sequence of subspaces. In addition, $P^T LP$ is also a graph Laplacian. This is useful because the coarse problem is often still too large to solve directly. Instead, one recursively applies a multigrid procedure to the coarse problem. That is, one smooths the coarse problem using Gauss-Seidel, and coarsens again.

A multigrid iteration, then, is a symmetric procedure:

1. Gauss-Seidel pass.
2. Coarse-grid correction (recursive multigrid step).
3. Reverse-direction Gauss-Seidel pass.

The result is a series of Gauss-Seidel passes on smaller and smaller problems until one reaches a problem small enough for a direct solve, followed by another series of Gauss-Seidel passes on larger and larger problems until the original problem is reached. This multigrid iteration is often called a *V-Cycle*.

Connection to Group-Synchronous Updates

We make a point here about the multigrid step that will be important later. Let us divide the nodes of the network into mutually exclusive subsets determined by the columns of P, and suppose we want to update each group of nodes simultaneously without converting to a coarse grid. That is, we will make an update

$$x_i \leftarrow x_i + \alpha \quad \forall i \in \mathcal{J}$$

where $\mathcal{J}$ is a set indices associated with the nonzero elements of some column of P. We do this by imposing a Galerkin condition

$$0 = \sum_{i \in \mathcal{J}} \left(s_i - \sum_{j=1}^{n} L_{ij} x_j - \alpha \sum_{k \in \mathcal{J}} L_{ik} \right) \tag{3.7}$$

and solving for α. We refer to this as a *group-synchronous update*. It is the group generalization of Equation (3.3).

We can rewrite the above condition in matrix form as

$$\begin{aligned} 0 &= \boldsymbol{e}_i^T P^T \left(\boldsymbol{s}_i - L\boldsymbol{x} - \alpha LP\boldsymbol{e}_i \right) \\ \implies \alpha \boldsymbol{e}_i^T (P^T LP) \boldsymbol{e}_i &= \boldsymbol{e}_i^T P^T \left(s_i - L\boldsymbol{x} \right). \end{aligned}$$

We now see that solving for α is equivalent to a Gauss-Seidel step in the coarse problem (3.6). Therefore, the multigrid iteration is equivalent to a series of group-synchronous updates, where groups are determined by the columns of P.

3.4 Nonlinear Gauss-Seidel

We now return to the nonlinear power flow problem. We first review a standard method for solving this problem, and then introduce the split Gauss-Seidel power flow and group-synchronous updates.

3.4.1 Classical Gauss-Seidel Power Flow

The classical nonlinear Gauss-Seidel method of solving the power flow equations, also sometimes called a Y-matrix method [64], is a standard power flow solution method. It is simple and has a small memory footprint, but by itself it converges slowly.

The method proceeds by iteratively cycling through each node in the network. At each node i, one does the following:

1. Compute the current injection given the current guess for the nodal voltage v_i:
$$I = s_i / v_i.$$

2. Considering I to be a target current injection, update v_i to match that target. That is, solve the equation
$$\bar{I} - \sum_{j=1}^{n} Y_{ij} v_j = 0 \tag{3.8}$$
for v_i. This results in the update formula
$$v_i \leftarrow \frac{\overline{s_i / v_i} - \sum_{j \neq i} Y_{ij} v_i}{Y_{ii}} = v_i + \frac{r_i}{Y_{ii}}, \tag{3.9}$$

where

$$r_i = \overline{s_i / v_i} - (Y\boldsymbol{v})_i \tag{3.10}$$

is the current injection residual.

At a PV node, only the real part of s_i is specified, so one first calculates the reactive power injection (the imaginary part of s_i) given the current voltage iterate $\boldsymbol{v}$. Then, because the voltage magnitude $|v_i^{sp}|$ is specified, one adjusts the voltage magnitude as a post-processing step. This gives us the three-step PV-node update

$$s_i \leftarrow P_i + \imath \operatorname{Im}\left(v_i \overline{(Y\boldsymbol{v})_i}\right) \tag{3.11}$$

$$v_i \leftarrow v_i + \frac{r_i}{Y_{ii}} \tag{3.12}$$

$$v_i \leftarrow |v_i^{sp}| \frac{v_i}{|v_i|} \tag{3.13}$$

At the slack node, one performs no update, as its voltage magnitude and phase angle are already specified.

Pseudocode for the classical Gauss-Seidel iteration is shown in Algorithm 2.

3.4.2 Split Gauss-Seidel Power Flow

In what follows, we will update groups of nodes simultaneously like in Section 3.3.2. A multigrid generalization of the above classical approach would force PQ and PV nodes to update their voltage magnitudes together, which we do not want as PV nodes have constant magnitude. For that reason, we present a different Gauss-Seidel method that splits each complex v_i into two real updates:

Algorithm 2: Classical Gauss-Seidel Iteration

procedure CLASSICALGAUSSSEIDELITERATION($\boldsymbol{v}, \boldsymbol{s}, Y$)
  for $i = 1, ..., n$ **do**
    if i is a PV node **then**
      $s_i \leftarrow P_i + \iota \operatorname{Im}\left(v_i \overline{(Y\boldsymbol{v})_i}\right)$ (Equation (3.11))
      $v_i \leftarrow v_i \frac{r_i}{Y_{ii}}$ (Equation (3.9) or (3.12))
      $v_i \leftarrow |v_i^{sp}| \frac{v_i}{|v_i|}$ (Equation (3.13))
    else if i is a PQ node **then**
      $v_i \leftarrow v_i \frac{r_i}{Y_{ii}}$ (Equation (3.9) or (3.12))
    end if
  end for
  return $\boldsymbol{v}$.
end procedure

one to update the voltage phase angle, and one to update the voltage magnitude.

Our voltage phase angle update proceeds in three steps:

1. Like in the classical approach above, we first compute our guess for the target current injection using Equation (3.8).
2. We set to zero the component of the current injection residual $r_i \in \mathbb{C}$ that voltage phase angles affect most strongly.
3. We solve for the phase angle of v_i that solves the equation from Step 2.

The voltage magnitude update is similar:

Algorithm 3: Split Gauss-Seidel Iteration

procedure CLASSICALGAUSSSEIDELITERATION($\boldsymbol{v}, \boldsymbol{s}, Y$)
  for $i = 1, ..., n$ **do**
    if i is a PV node **then**
      $s_i \leftarrow P_i + \imath \operatorname{Im}\left(v_i \overline{(Y\boldsymbol{v})_i}\right)$ (Equation (3.11))
      Update v_i using Equations (3.17) and (3.18)
    else if i is a PQ node **then**
      Update v_i using Equations (3.17) and (3.18)
      Update v_i using Equations (3.20) and (3.21)
    end if
  end for
  return $\boldsymbol{v}$.
end procedure

1. Compute our guess for the target current injection using Equation (3.8).
2. Set to zero the component of the current injection residual $r_i \in \mathbb{C}$ that voltage magnitudes affect most strongly.
3. We solve for the magnitude of v_i that solves the equation from Step 2.

To determine which components of the current residual to use for these two updates, we make the physical approximation here that all power lines in a network are of the same type; that is, they are constructed of similar material, have similar cross-sections, and therefore have similar electrical properties. However, they have different lengths. This means that the electrical admittances of the power lines have similar complex phase angles, but different magnitudes, as magnitude is determined by line length [30]. In addition, we neglect trans-

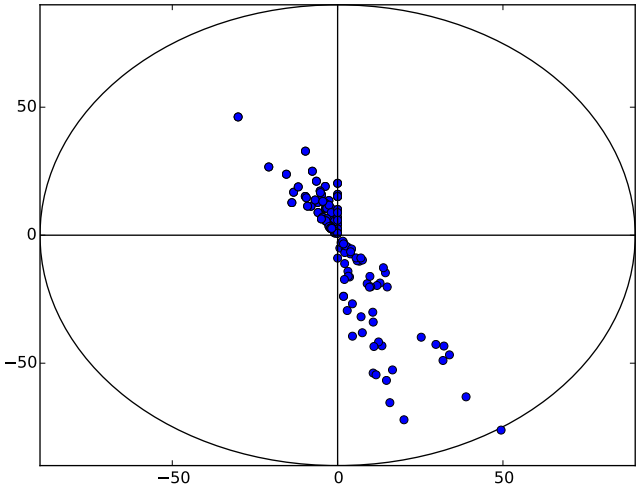


Figure 3.1: Nonzero elements of IEEE 57-bus admittance matrix.

formers and shunt admittances.

This physical approximation results in the mathematical approximation of the admittance matrix

$$Y \approx zL, \tag{3.14}$$

where L is a real-valued graph Laplacian matrix plus some positive diagonal elements for shunt admittances, and $z \in \mathbb{C}$ is a scalar complex rotation ($|z| = 1$). The scalar z is a property of the common power line type, while L is due to the lengths of the various power lines. One can see this phenomenon in Figure 3.1, which shows the admittance values of the entries in the admittance matrix of the standard IEEE 57-bus test network.

In practice, we compute z simply by taking the mean of the diagonal of Y.

Using this approximation and the small-angle approximation from Section

3.3.1, we write

$$\begin{aligned}\overline{zv_i}r_i \approx \overline{z}\,(\overline{s_i} - \overline{v_i}(zL\boldsymbol{v})_i) =& \overline{zs_i} - \overline{v_i}(L\boldsymbol{v})_i \\ =& \overline{zs_i} - \sum_{j=1}^{n} L_{ij}|v_i||v_j|e^{\iota(\theta_j-\theta_i)} \\ \approx& \overline{zs_i} - \sum_{j=1}^{n} L_{ij}|v_i||v_j|\left(1 + \iota(\theta_j - \theta_i)\right). \end{aligned} \tag{3.15}$$

Taking real and imaginary components, we see that

$$\begin{aligned}\mathrm{Re}\,(\overline{zv_i}r_i) &\approx \mathrm{Re}\,(\overline{zs_i}) - \sum_{j=1}^{n} L_{ij}|v_i||v_j| \\ \mathrm{Im}\,(\overline{zv_i}r_i) &\approx \mathrm{Im}\,(\overline{zs_i}) - \sum_{j=1}^{n} L_{ij}|v_i||v_j|(\theta_j - \theta_i).\end{aligned}$$

Thus, under these approximations, voltage magnitudes most strongly affect the real component of $\overline{zv_i}r_i$ and voltage phase angles most strongly affect the imaginary component of $\overline{zv_i}r_i$. These are the components of r_i that we use.

We update separately a node's voltage phase angle and magnitude by setting the imaginary and real components of $\overline{zv_i}r_i$, respectively, to zero, and updating v_i to match that component of the current injection. Mathematically, we write this in the following way:

- To update the phase angle of v_i by an amount $\lambda \in \mathbb{R}$, we set

$$\mathrm{Im}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j - \overline{v_i}Y_{ii}v_i e^{\iota\lambda}\right)\right] = 0. \tag{3.16}$$

  Solving for $e^{\iota\lambda}$ gives two possibilities:

$$e^{\iota\lambda} = \frac{\iota c \pm \sqrt{|x|^2 - c^2}}{x}, \tag{3.17}$$

  where

$$x = \overline{z}|v_i|^2 Y_{ii} \qquad c = \mathrm{Im}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j\right)\right].$$

We select the option that causes the smallest angle change.

We then update v_i with the rotation

$$v_i \leftarrow v_i e^{i\lambda}. \tag{3.18}$$

- To update the voltage magnitude of v_i by a scaling $\alpha \in \mathbb{R}^+$, we set

$$\mathrm{Re}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j - \overline{v_i}Y_{ii}(\alpha v_i)\right)\right] = 0. \tag{3.19}$$

Solving for α gives us

$$\alpha = \frac{\mathrm{Re}\left[\overline{z}(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j)\right]}{\mathrm{Re}[\overline{z}Y_{ii}|v_i|^2]} = 1 + \frac{\mathrm{Re}[\overline{zv_i}r_i]}{\mathrm{Re}[\overline{z}Y_{ii}|v_i|^2]}. \tag{3.20}$$

We then update v_i with the scaling

$$v_i \leftarrow \alpha v_i. \tag{3.21}$$

We show the split Gauss-Seidel algorithm in Algorithm 3. Notice the similarities and differences between that and the classical Gauss-Seidel iteration in Algorithm 2.

PV nodes. At a PV node the voltage magnitude is specified, so we skip the voltage magnitude update. Additionally, s_i is not fully specified at a PV nodes, so we must first update our guess of the reactive power injection using Equation (3.11).

No solution to angle update. It is possible that there are no solutions to Equation (3.16). This occurs if $|x|^2 < c^2$. If there are no solutions, we simply minimize the residual by choosing $e^{\imath\lambda}$ to either maximize or minimize $\mathrm{Im}[\overline{zv_i}Y_{ii}v_i e^{i\lambda}]$, whichever is appropriate.

Updating with the small-angle approximation. We typically expect phase angles to only change by small amounts. Therefore, in solving for the phase angle update, one may make the approximation $e^{\imath\lambda} \approx 1 + \imath\lambda$. In this case, solving for λ gives the solution

$$\lambda = \frac{\mathrm{Im}[\overline{zv_i}r_i]}{\mathrm{Im}[\imath\bar{z}Y_{ii}|v_i|^2]} \tag{3.22}$$

and the update formula

$$v_i \leftarrow (1 + \imath\lambda)v_i. \tag{3.23}$$

This update formula is less expensive than Equation (3.17), and its use requires no branching. Note that, in this case, the voltage magnitude of a PV node is perturbed somewhat, and so it must be corrected using Equation (3.13).

3.4.3 Group-Synchronous Updates

We can extend the above method by updating groups of nodes together, as in Section 3.3.2. Group-synchronous updates allow us to change how a set of nodes share power with its surroundings while keeping power sharing within the group approximately the same.

In particular, suppose we wish to make the group phase angle update

$$v_i \leftarrow v_i e^{\imath\lambda} \quad \forall i \in \mathcal{J}, \tag{3.24}$$

where $\mathcal{J}$ is some subset of node indices. Here the scalar λ controls the update for all nodes in $\mathcal{J}$. To determine the value of λ, we impose a condition like in Equation (3.16):

$$\mathrm{Im}\left[\overline{z}\sum_{i\in\mathcal{J}}\overline{v_i}\left(\overline{s_i/v_i} - \sum_{j\notin\mathcal{J}} Y_{ij}v_j - e^{\imath\lambda}\sum_{k\in\mathcal{J}} Y_{ik}v_k\right)\right] = 0. \tag{3.25}$$

This is analogous to the linear Galerkin condition in Equation (3.7). Solving for $e^{\imath\lambda}$ gives

$$e^{\imath\lambda} = \frac{\imath c \pm \sqrt{|x|^2 - c^2}}{x} \tag{3.26}$$

where

$$x = \overline{z} \sum_{i,k \in \mathcal{J}} \overline{v_i} Y_{ik} v_k \qquad c = \operatorname{Im}\left[\overline{z} \sum_{i \in \mathcal{J}} \left(\overline{s_i} - \overline{v_i} \sum_{j \notin \mathcal{J}} Y_{ij} v_j \right)\right].$$

Similarly, suppose we want to make the group magnitude update

$$v_i \leftarrow \alpha v_i \quad \forall i \in \mathcal{J}. \tag{3.27}$$

We impose a condition like in Equation (3.21):

$$\operatorname{Re}\left[\overline{z} \sum_{i \in \mathcal{J}} \overline{v_i} \left(\overline{s_i / v_i} - \sum_{j \notin \mathcal{J}} Y_{ij} v_j - \alpha \sum_{k \in \mathcal{J}} Y_{ik} v_i \right)\right] = 0. \tag{3.28}$$

Solving for α gives us

$$\alpha = \frac{\operatorname{Re}\left[\overline{z} \sum_{i \in \mathcal{J}} (\overline{s_i} - \overline{v_i} \sum_{j \notin \mathcal{J}} Y_{ij} v_j)\right]}{\operatorname{Re}[\overline{z} \sum_{i,k \in \mathcal{J}} \overline{v_i} Y_{ik} v_k]} = 1 + \frac{\operatorname{Re}[\overline{z} \sum_{i \in \mathcal{J}} \overline{v_i} r_i]}{\operatorname{Re}[\overline{z} \sum_{i,k \in \mathcal{J}} \overline{v_i} Y_{ik} v_k]}. \tag{3.29}$$

Note the following:

- A node can appear in more than one group, and singleton groups are possible (which simply reduce to the one-variable split Gauss-Seidel update of Section 3.4.2).
- It is possible to have both PQ and PV nodes in a single group. In this case, we exclude the PV nodes from the voltage magnitude update step.

Algorithm 4 shows the pseudocode for a generalization of the split Gauss-Seidel iteration of Algorithm 3 that takes a list of groups and computes a group-synchronous update for each group.

Algorithm 4: Gauss-Seidel Iteration

function ANGLEUPDATE($\boldsymbol{v}, \boldsymbol{s}, Y, \mathcal{I}$)
  for each $i \in \mathcal{I}$ that is a PV node **do**
    $s_i \leftarrow P_i + \imath \operatorname{Im}\left(v_i \overline{(Y\boldsymbol{v})_i}\right)$ (Equation (3.11))
  end for
  Calculate λ using Equation (3.26).
  for all $i \in \mathcal{I}$ **do**
    Update v_i using Equation (3.18)
  end for
end function
function MAGNITUDEUPDATE($\boldsymbol{v}, \boldsymbol{s}, Y, \mathcal{I}$)
  Calculate α using Equation (3.29).
  for all $i \in \mathcal{I}$ **do**
    Update v_i using Equation (3.21).
  end for
end function
procedure GAUSSSEIDELITERATION($\boldsymbol{v}, \boldsymbol{s}, Y$, groups_list)
  for $\mathcal{J}$ in groups_list **do**
    $\mathcal{J}_{\mathcal{PQ}} \leftarrow [i \text{ for } i \in \mathcal{J} \text{ if } i \text{ is a PQ node}]$.
    ANGLEUPDATE($\boldsymbol{v}, \boldsymbol{s}, Y, \mathcal{J}$).
    MAGNITUDEUPDATE($\boldsymbol{v}, \boldsymbol{s}, Y, \mathcal{J}_{\mathcal{PQ}}$).
  end for
  return $\boldsymbol{v}$.
end procedure

Selection of Groups

To use the group-synchronous approach, we need a way to choose sets of nodes to update together. We choose update groups based on the observation that nodees connected by lines with high admittance tend to shift state together. That is, if nodes i and j are connected by a high-admittance line and the voltage at node i changes, either node j's voltage will shift with it, or the power transfer between nodees i and j will itself shift dramatically. This is similar to the reasoning for node aggregation selection in standard algebraic multigrid [71].

Recall from Section 3.3.2 that the linear multigrid framework recursively coarsens the problem to create a hierarchy of problems of different sizes. Section 3.3.2 shows that this multigrid hierarchy is equivalent to performing group-synchronous updates on the original problem using a hierarchy of nested groups. Having such a range of group sizes allows one to more effectively capture different types of error modes (see, e.g. [14]). We want a range of group sizes to capture the same type of benefits seen in the linear group-synchronous case. As with linear multigrid, we define these hierarchically as groups of groups.

First, we define a set of "next-smallest" groups after the singletons. We order nonzero upper-triangular elements of the admittance matrix Y by the magnitude of the matrix elements. This gives us an ordered list of (i, j, g) tuples, where i and j are nodes and g is the magnitude of the admittance of the line connecting them. We then apply Algorithm 6.1 of [72]: We place all nodes in groups by iterating through the ordered list twice. In the first iteration, for each tuple, if i and j have not yet been placed in any group, they are placed in a new group together. In the second iteration, at least one of i and j has already been

placed in a group. If either i or j has not yet been placed in any group, then it is placed in the group of the other. The result is a disjoint set of node groups to be used in group-synchronous updates. We show pseudocode for this method in Algorithm 5.

Having run Algorithm 5, we have a disjoint set of groups B with n_2 groups, where typically $n/5 < n_2 < n/2$. We now show how to extend this to create a hierarchy of nested groups. It is a bootstrapping process that is similar to those used in the linear algebraic multigrid context.

Define a matrix $P \in \{0,1\}^{n \times n_2}$ such that each row of P corresponds to a network node, each column corresponds to a group, and

$$P_{ij} = \begin{cases} 1 & \text{node } i \text{ is in group } j \\ 0 & \text{otherwise} \end{cases} . \tag{3.30}$$

Given this, we define a *preprocessing coarse grid* $\tilde{Y}_2 = P^T Y P$ on which we can then run Algorithm 5 again to obtain larger groups of nodes. Applying this procedure recursively gives us a full hierarchy of nested groups.

Algorithm 6 shows pseudocode for this procedure. The level-h groups are groups of level-$h-1$ groups seen in the columns of P_h. In terms of the original network, the level-h groups are columns of the matrix $\tilde{P}_h = P_1 \cdots P_h \in \{0,1\}^{n \times n_h},\ h = 1, ..., H$.

Note the following:

- $P_1 = I$. Thus, level-1 groups are simply the original set of nodes.
- Each node appears in exactly one level-h group for each h. Thus, for each $h' < h$, the set of level-h groups are a partition of the level-h' groups.

Algorithm 5: Aggregation of Nodes

procedure AGGREGATE(Y)
    From Y, create list of strictly upper triangular nonzero elements L = list($(i, j, |Y[i, j]|)$).
    Order L in descending order by $|Y[i, j]|$.
    Initialize list of groups $B = \{\}$.
    for (i, j, m) in L **do**
        if *i.group* and *j.group* are not yet set **then**
            Create new group $b = \{i, j\}$.
            $B \leftarrow B \bigcup \{b\}$.
            $i.group \leftarrow b, \quad j.group \leftarrow b$.
        end if
    end for
    for (i, j, m) in L **do**
        if *i.group* not yet set **then**
            $j.group \leftarrow j.group \cup \{i\}$.
            $i.group \leftarrow j.group$.
        else if *j.group* not yet set **then**
            $i.group \leftarrow i.group \cup \{j\}$.
            $j.group \leftarrow i.group$.
        end if
    end for
    return B.
end procedure

Algorithm 6: Construct Hierarchy

procedure CONSTRUCTHIERARCHY(Y, min_size)
    $\tilde{Y} \leftarrow Y$.
    $h = 1$.
    while size($\tilde{Y}$) < min_size **do**
        $B \leftarrow$ AGGREGATE($\tilde{Y}$)
        Construct P_h as in Equation (3.30).
        $\tilde{Y} \leftarrow P_h^T \tilde{Y} P_h$.
        $h \leftarrow h + 1$.
    end while
    $H \leftarrow h$.
    return $P_1, ..., P_H$.
end procedure

3.5 Multigrid

While the above method enables new kinds of Gauss-Seidel updates, the cost of an update scales linearly with the number of nodes to be updated together. We can reduce the cost of these updates so that each one incurs amortized constant cost. Recall from Section 3.3.2 that in the linear case group-synchronous updates are equivalent to multigrid cycles. Here we show how to construct a nonlinear multigrid cycle for the power flow equations that is equivalent to the group-synchronous updates of Section 3.4.3. We begin by supposing that there are no PV nodes in the network. We show in Section 3.5.3 how to incorporate PV nodes.

The conditions of Equations (3.25) and (3.28) requires we choose our update so that although there may be errors within the group, a component of the average power injection is correct. Let $\mathcal{J}_j$ denote the nonzero indices of the j'th column of $\tilde{P}_h$, and consider the matrix $\mathcal{I} = YD_v\tilde{P}_h$. Element $\mathcal{I}_{ij}$ represents the contribution to the current injection at node i due to the set of nodes $\mathcal{J}_j$. Then element (i, j) of the matrix

$$\begin{aligned} Y(\boldsymbol{v}, h) =& \tilde{P}_h^T \overline{D_v} \mathcal{I} \\ =& \tilde{P}_h^T \overline{D_v} Y D_v \tilde{P}_h \end{aligned} \tag{3.31}$$

represents the contribution to the sum of power injections at node set $\mathcal{J}_i$ due to the node set $\mathcal{J}_j$. Note that the matrix $Y(\boldsymbol{v}, h) \in \mathbb{C}^{n_h}$ is significantly smaller than the original admittance matrix, and so information about sums of power injections and the effects of multiplicative updates is compactly represented. We can leverage this compact representation to create a faster iteration.

3.5.1 Coarse Power Flow

We first present a lemma.

Lemma 1. *Let $P \in \{0, 1\}^{n \times \ell}$, $\ell \leq n$ denote a matrix with exactly one 1 in each row, and let $\boldsymbol{x} \in \mathbb{C}^{\ell}$. Then*

$$P^T diag(P\boldsymbol{x}) = diag(\boldsymbol{x})P^T$$

Proof. Assume without loss of generality that the rows of U are ordered by the

columns of their nonzero elements; that is, assume that

$$P^T = \begin{bmatrix} 1 & \cdots & 1 & & & & & & \\ & & & 1 & \cdots & 1 & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \cdots 1 \end{bmatrix}.$$

Then

$$P\boldsymbol{x} = \begin{bmatrix} x_1 & \cdots x_1 & \cdots & x_\ell & \cdots & x_\ell \end{bmatrix}^T,$$

and so

$$P^T \operatorname{diag}(U\boldsymbol{x}) = \begin{bmatrix} x_1 & \cdots & x_1 & & & & & & \\ & & & x_2 & \cdots & x_2 & & & \\ & & & & & & \ddots & & \\ & & & & & & & x_\ell & \cdots x_\ell \end{bmatrix} = \operatorname{diag}(\boldsymbol{x})P^T. \tag{3.32}$$

□

Now, let $\boldsymbol{v}_0$ denote a voltage vector. Suppose we perform a sequence of level-h group-synchronous updates, let $\boldsymbol{\alpha} \in \mathbb{R}^{n_h}$ denote the vector of magnitude updates, and let $e^{\imath\boldsymbol{\lambda}} \in \mathbb{C}^{n_h}$ denote the vector of phase angle updates, so that

$$\boldsymbol{v} = D_{\boldsymbol{v}_0} \tilde{P}_h \left(\boldsymbol{\alpha} \circ e^{\imath\boldsymbol{\lambda}} \right), \tag{3.33}$$

where the $\circ$ operator represents the Hadamard product. We wish to update the i'th level-h group, that is, the nonzero elements of $\tilde{P}_h \boldsymbol{e}_i$. The following theorem lets us write this update condition in terms of a coarse power flow problem.

Theorem 2. *Let*

$$\boldsymbol{s}^h = \tilde{P}_h^T \boldsymbol{s} \in \mathbb{C}^{n_h} \tag{3.34}$$

$$\boldsymbol{v}^h = \boldsymbol{\alpha} \circ e^{\imath\boldsymbol{\lambda}} \in \mathbb{C}^{n_h} \tag{3.35}$$

$$Y(\boldsymbol{v}, h) = \tilde{P}_h^T \overline{D_{\boldsymbol{v}}} Y D_{\boldsymbol{v}} \tilde{P}_h \in \mathbb{C}^{n_h \times n_h}. \tag{3.36}$$

Then the group-synchronous phase angle update condition of Equation (3.25) *is satisfied if and only if*

$$0 = \operatorname{Im}\left(\overline{z}\left[\overline{s_i^h} - \overline{v_i^h}\sum_{j\neq i} Y_{ij}(\boldsymbol{v}_0, h)v_j^h - \overline{v_i^h}Y_{ii}(\boldsymbol{v}_0, h)v_i^h e^{\iota\lambda_i}\right]\right) \tag{3.37}$$

and the group-synchronous magnitude update condition of Equation (3.28) *is satisfied if and only if*

$$0 = \operatorname{Re}\left(\overline{z}\left[\overline{s_i^h} - \overline{v_i^h}\sum_{j\neq i} Y_{ij}(\boldsymbol{v}_0, h)v_j^h - \overline{v_i^h}Y_{ii}(\boldsymbol{v}_0, h)(\alpha_i v_i^h)\right]\right). \tag{3.38}$$

Proof. We prove the result for the phase angle update of Equation (3.37). Suppose first that $\boldsymbol{v}^h = \mathbf{1}_{n_h}$. From Equation (3.25) we have

$$\begin{aligned}
0 &= \operatorname{Im}\left(\overline{z}\sum_{\ell\in\mathcal{J}_i}^{m}\left[s_\ell - \overline{v_\ell}\sum_{k\notin\mathcal{J}_i} Y_{\ell k}v_k - \overline{v_\ell}\sum_{k\in\mathcal{J}_i} Y_{\ell k}v_k e^{\iota\lambda}\right]\right) \\
&= \operatorname{Im}\left(\overline{z}\sum_{\ell\in\mathcal{J}_i}^{m}\left[s_\ell - \overline{v_\ell}\sum_{k=1}^{n} Y_{\ell k}v_k - \overline{v_\ell}\sum_{k\in\mathcal{J}_i} Y_{\ell k}v_k(e^{\iota\lambda} - 1)\right]\right) \\
&= \operatorname{Im}\left(\overline{z}(\tilde{P}_h\boldsymbol{e}_i)^T\left[\overline{\boldsymbol{s}} - \overline{D_{\boldsymbol{v}}}Y\boldsymbol{v} - (e^{\iota\lambda} - 1)\overline{D_{\boldsymbol{v}}}YD_{\boldsymbol{v}}\tilde{P}_h\boldsymbol{e}_i\right]\right) \\
&= \operatorname{Im}\left(\overline{z}\boldsymbol{e}_i^T\left[\overline{\boldsymbol{s}^h} - Y(\boldsymbol{v}, h)\mathbf{1}_{n_h} - (e^{\iota\lambda} - 1)Y(\boldsymbol{v}, h)\boldsymbol{e}_i\right]\right) \\
&= \operatorname{Im}\left(\overline{z}\left[\overline{s_i^h} - \overline{v_i^h}\sum_{j\neq i} Y_{ij}(\boldsymbol{v}_0, h)v_j^h - \overline{v_i^h}Y_{ii}(\boldsymbol{v}_0, h)v_i^h e^{\iota\lambda_i}\right]\right).
\end{aligned} \tag{3.39}$$

It remains to be shown that the equivalence of Equations (3.25) and (3.37) hold after we update $\boldsymbol{v}^h$. So, assume that $\boldsymbol{v}^h \neq \mathbf{1}_{n_h}$. Suppose we update $\boldsymbol{v}_0$ to $\boldsymbol{v}$ using Equation (3.33):

$$\boldsymbol{v} = D_{\boldsymbol{v}_0}\tilde{P}_h\left(\boldsymbol{\alpha}\circ e^{\iota\boldsymbol{\lambda}}\right) = D_{\boldsymbol{v}_0}\tilde{P}_h\boldsymbol{v}^h.$$

Two applications of Lemma 1 allow us to write

$$\begin{aligned} Y(\boldsymbol{v},h) = \tilde{P}_h^T \overline{D_{\boldsymbol{v}}} Y D_{\boldsymbol{v}} \tilde{P}_h &= \tilde{P}_h^T \overline{D_{D_{\boldsymbol{v}_0}\tilde{P}_h \boldsymbol{v}^h}} Y D_{D_{\boldsymbol{v}_0}\tilde{P}_h \boldsymbol{v}^h} \tilde{P}_h \\ &= \tilde{P}_h^T \overline{D_{\tilde{P}_h \boldsymbol{v}^h}} D_{\boldsymbol{v}_0} Y D_{\tilde{P}_h \boldsymbol{v}^h} D_{\boldsymbol{v}_0} \tilde{P}_h \\ &= \overline{D_{\boldsymbol{v}^h}} \tilde{P}_h^T \overline{D_{\boldsymbol{v}_0}} Y D_{\boldsymbol{v}_0} \tilde{P}_h D_{\boldsymbol{v}^h} \\ &= \overline{D_{\boldsymbol{v}^h}} Y(\boldsymbol{v}_0, h) D_{\boldsymbol{v}^h}. \end{aligned}$$

So, Equation (3.39) is now

$$\begin{aligned} 0 &= \operatorname{Im}\left(\bar{z}\boldsymbol{e}_i^T \left[\overline{\boldsymbol{s}^h} - \overline{D_{\boldsymbol{v}^h}} Y(\boldsymbol{v},h) D_{\boldsymbol{v}^h} \mathbf{1}_{n_h} - (e^{\iota\lambda} - 1)\overline{D_{\boldsymbol{v}^h}} Y(\boldsymbol{v},h) D_{\boldsymbol{v}^h} \boldsymbol{e}_i\right]\right) \\ &= \operatorname{Im}\left(\bar{z}\left[\overline{s_i^h} - \overline{v_i^h} \sum_{j \neq i} Y_{ij}(\boldsymbol{v}_0,h) v_j^h - \overline{v_i^h} Y_{ii}(\boldsymbol{v}_0,h) v_i^h e^{\iota\lambda_i}\right]\right). \end{aligned}$$

The proof for the magnitude update is similar. □

The above theorem shows that an update on level-h group i is equivalent to a one-variable update on the *coarse power flow problem*

$$D_{\boldsymbol{v}^h} \overline{Y(\boldsymbol{v}_0,h)\boldsymbol{v}^h} = \boldsymbol{s}^h. \tag{3.40}$$

In the context of the level-h coarse power flow problem, we refer to the level-h groups as *level-h coarse nodes* or simply *coarse nodes.* Note that $\boldsymbol{v}^h = \mathbf{1}_{n_h}$ when the coarse problem is first constructed, and that if $\boldsymbol{v}$ solves the original problem, then $\boldsymbol{v}^h = \mathbf{1}_{n_h}$ solves the coarse problem.

If we solve the coarse power flow problem either partially (e.g. with a few steps of an iterative method) or fully, we can pass the result back to the original fine level with

$$\boldsymbol{v} \leftarrow D_{\boldsymbol{v}_0} \tilde{P}_h \boldsymbol{v}^h.$$

In what follows, we will write Y^h for $Y(\boldsymbol{v},h)$ when the intended vector $\boldsymbol{v}$ is clear.

3.5.2 Recursion

Algorithm 4 requires a groups_list parameter, an ordered list of groups through which to cycle. We construct groups_list by first using Algorithm 6 to construct $\tilde{P}_1, ..., \tilde{P}_h$. Our groups_list is then the following

$$\begin{aligned}\text{init\_list} =& \text{nnz}(\tilde{P}_1[:,1]), ..., \text{nnz}(\tilde{P}_1[:,n]), \text{nnz}(\tilde{P}_2[:,1]), ..., \text{nnz}(\tilde{P}_2[:,n_2]),\\ & ..., \text{nnz}(\tilde{P}_H[:,1]), ..., \text{nnz}(\tilde{P}_H[:,n_H])\\ \text{groups\_list} =& \text{init\_list}, \text{reversed}(\text{init\_list}).\end{aligned}$$

That is, we traverse up through the hierarchy, from the finest nodes up to the coarsest, and then down from the coarsest back to the finest in reverse order. This symmetric update pattern is the well-known V-Cycle [14]. As in the linear multigrid case, we can exploit the hierarchical nature of the V-Cycle to reduce its cost.

We will construct a nested series of coarse problems. For each problem, we will apply a split Gauss-Seidel iteration before moving on to the next. Due to Theorem 2, the result of the V-cycle pattern will be the same as if we had applied the associated set of group-synchronous updates. In multigrid terminology, the application of the Gauss-Seidel iteration to a given problem is *smoothing*, and Gauss-Seidel is the *smoother*.

To construct the V-cycle update, we apply the smoother to level 1 to obtain a voltage vector $\boldsymbol{v}$, construct the level-2 coarse problem (3.40) via Equations (3.34)–(3.36), and then apply the smoother to the level-2 coarse problem. Given the group order of the V-Cycle, we now shift to level-3 groups. We could pass the level-2 updates to level 1 with the update

$$\boldsymbol{v} \leftarrow D_v P_2 \boldsymbol{v}^2 \tag{3.41}$$

and then construct the level-3 coarse problem via Equations (3.34)–(3.36) with $h = 3$. However, we develop a faster method by delaying the update to the original voltage vector and constructing the level-3 problem directly from level 2 with

$$\boldsymbol{s}^3 = P_3^T \boldsymbol{s}^2 \qquad Y^3 = P_3^T \overline{D_{\boldsymbol{v}^2}} Y^2 D_{\boldsymbol{v}^2} P_3. \tag{3.42}$$

By an argument similar to the proof of Theorem 2, the result is the same as if we had used Equations (3.41) and (3.34)–(3.36) to construct the level-3 coarse problem from level 1.

For each level h, then, we apply the smoother, and then directly construct the level-$h+1$ coarse problem. We recurse in this way through the entire hierarchy until we reach the top at level H. Once we apply our smoother to level H, we must then pass the voltage vector updates back down the hierarchy. This is done in sequence as follows:

$$\begin{aligned}
\boldsymbol{v}^{H-1} &\leftarrow D_{\boldsymbol{v}^{H-1}} P_H \boldsymbol{v}^H \\
\boldsymbol{v}^{H-2} &\leftarrow D_{\boldsymbol{v}^{H-2}} P_{H-1} \boldsymbol{v}^{H-1} \\
&\vdots \\
\boldsymbol{v} &\leftarrow D_{\boldsymbol{v}} P_2 \boldsymbol{v}^2,
\end{aligned}$$

applying the smoother (in reverse order) at each level of the hierarchy before passing the update to the next-finer level. The result is the output of a V-Cycle iteration, and is identical to the result we would receive if we had applied the corresponding group-synchronous updates to the level-1 problem in V-cycle order.

Note that in each V-Cycle we need to construct a new coarse matrix Y^h, as the voltage vectors used to construct them change over time. However, the

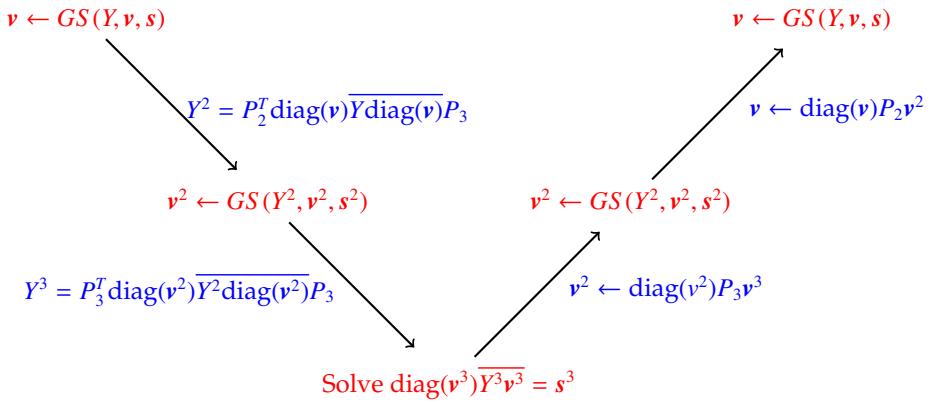


Figure 3.2: Power Flow V-Cycle. A Gauss-Seidel smoother is applied at each level before passing the problem to a coarser level. Once a sufficiently small problem is attained, it can be solved directly before passing the problem back to the finer levels for more Gauss-Seidel smoothing.

nonzero patterns of Y^h do not change over time, and so memory for them can be preallocated and reused throughout the computation. Furthermore, the coarse target vectors s^h do not change, and so can be precomputed.

In summary, our algorithm recursively applies a smoother at a level h, then based on the current iterate v^h, constructs a level-$h+1$ coarse power flow problem. Once the coarsest level of the hierarchy is reached, the method travels back down the hierarchy, smoothing and passing the cumulative v^h's back down to finer levels. This procedure is the classic V-Cycle. The algorithm for the V-Cycle is shown in Algorithm 7, and a diagram is shown in Figure 3.2.

Full Approximation Scheme. A well-known general method for performing nonlinear multigrid is the Full Approximation Scheme (or FAS) [14]. It is worth noting that the method described in this paper is not a use of FAS. The FAS requires a hierarchy of coarse problems that remain constant throughout processing, and passes information through the hierarchy via a changing target

vector. In contrast, our method is a result of accelerating group-synchronous updates, and passes information through the admittance matrices rather than the target vectors.

3.5.3 Incorporating PV Nodes

Prior to this point in Section 3.5, we assumed that there were no PV nodes in the network. One option would be to simply disallow PQ and PV nodes to update together. However, this breaks the underlying philosophy of group-synchronous updates, which is to allow many buses to update together, so we take another approach.

Note that the group-synchronous updates of Section 3.4.3 and Algorithm 4 allow PQ and PV nodes to update together, and simpy do not include PV nodes in the voltage magnitude portion of the update. Here we take the same approach, but do so with groups of size at most two.

In particular, suppose Algorithm 6 returns $P_1, ..., P_H$, where $P_1 = I$. Beginning with P_2, we *type separate*: split any groups that contain more than one type of node so that each group consists of only one type. That is, we factor P_2 into $P_2 = P_2' P_2''$, where $P_2' \in \{0,1\}^{n \times n_2'}$ and $P_2'' \in \{0,1\}^{n_2' \times n_2}$ where $n_2' > n_2$. The matrix P_2' defines our level-2 coarse nodes and is used in the construction of Y^2. The columns of P_2'' describe the sets of coarse nodes that were combined prior to type separation, and we refer to these groups of coarse nodes as *update sets*.

When smoothing, we do not perform single-node updates on the coarse nodes as in Sections 3.4.1 and 3.5.2. Instead, we use the update sets of P_h'' to

define group-synchronous updates on the level-2 coarse problem, as is done in Section 3.4.3. However, these update sets are always of size at most two, with one PQ and one PV coarse node, so their group-synchronous updates only incur constant cost. Note that this is not a group-synchronous update with the groups of level-1 nodes defined by the columns of $P_2 = \tilde{P}_2$, but is a group-synchronous update with groups of level-2 nodes.

Once a level-h coarse power flow problem is defined in terms of P'_h, we must update P_{h+1} so that it has $n_{h'}$ rows and properly aggregates the level-h node into level-h + 1 groups. Once this is done, we split the level-h + 1 groups in the same way and continue working our way up the hierarchy. The result is the matrices $P'_2, ..., P'_H$ and $P''_2, ..., P''_H$ that define the coarse nodes and the update sets, respectively.

This approach allows us to efficiently update PQ and PV nodes together without violating the different constraints imposed by different node types.

3.5.4 Variants

Full Multigrid. An important method related to the V-Cycle is the Full Multigrid Cycle or FMG-Cycle [14]. This method starts at the top of the hierarchy, performing V-cycles starting at each level as it travels down the hierarchy. This method is useful for cheaply solving "big picture" parts of the problem first, and is often highly effective while only about twice as expensive as the V-cycle. Pseudocode for the FMG-Cycle is shown in Algorithm 8.

Coarse Newton's Method. The above methods do not require that we actually use the full hierarchy; we could "cut off" a V-Cycle at level h, not performing those updates of levels above h. This is usually not done because those are the steps that move information between the most distant nodes in a network and the steps themselves are extremely computationally inexpensive.

However, one option is to cut off a V-Cycle at level h and fully or partly solve the coarse power flow problem at that level with a Newton or quasi-Newton method. Because Newton and quasi-Newton methods invert or partly invert network matrices, they transfer information between distant nodes. On the other hand, because the level-h problem is so much smaller than the original one, matrix factorization is much cheaper.

Using a Jacobi Smoother. A Jacobi smoother makes all the same updates that a Gauss-Seidel smoother does, but makes those updates simultaneously rather than sequentially. The Jacobi iteration is popular in linear multigrid methods because it is highly parallelizable, but there is a tradeoff with stability and convergence. When using Jacobi in a multigrid method for a linear system of equations with a Jacobi smoother, one needs to *weight* or *damp* the Jacobi step to obtain convergence. That is, after computing a possible update $\tilde{\boldsymbol{v}}$, one then sets

$$\boldsymbol{v} \leftarrow \omega\tilde{\boldsymbol{v}} + (1-\omega)\boldsymbol{v}$$

for some $\omega \in (0, 1)$. The optimal value of ω varies from problem to problem, but a selection of $\omega = 2/3$ is often effective [14].

As in the linear case, using a Jacobi smoother in multigrid power flow is less stable than a Gauss-Seidel smoother. We found experimentally that the weighting factor ω needed when the residual $\boldsymbol{r}$ is large is often much smaller

than the ω needed when it is small. So, we use a *weighting schedule* with Jacobi iterations that allows ω to grow as the algorithm progresses. In particular, we define a set of bounds $b_1, ..., b_k, b_1 = 0, b_k = \infty$ and a set of weights $\omega_1, ..., \omega_{k-1}$. Then, if $b_i < \|\boldsymbol{r}\|_\infty \leq b_{i+1}$, we set $\omega = \omega_i$.

We leave the question of how best to choose this schedule as future work.

Algorithm 7: V-Cycle Iteration

---

procedure V_CYCLE($\boldsymbol{v}, \boldsymbol{s}, Y$, P_list)
  $\boldsymbol{v} \leftarrow$ GAUSSSEIDELITERATION($\boldsymbol{v}, \boldsymbol{s}, Y$)
  if P_list is not empty **then**
    $P \leftarrow$ P_list[0].
    $Y_C \leftarrow \overline{P^T \text{diag}(\boldsymbol{v})} Y \text{diag}(\boldsymbol{v}) P$.
    $\boldsymbol{s}_C \leftarrow P^T \boldsymbol{s}$.
    $\boldsymbol{v}_C \leftarrow$ V_CYCLE($\mathbf{1}, \boldsymbol{s}_C, Y_C$, P_list[1:]).
    $\boldsymbol{v} \leftarrow D_v P \boldsymbol{v}_C$.
  end if
  $\boldsymbol{v} \leftarrow$ GAUSSSEIDELITERATION($\boldsymbol{v}, \boldsymbol{s}, Y$, reverse = True)
  return $\boldsymbol{v}$.
end procedure

3.6 Analysis

In this section, we analyze the convergence of Algorithm 3 and the two-grid version of Algorithm 7 (that is, the variant in which we switch to a Newton solve at coarse level $h = 2$). We will first develop a local proof of convergence

Algorithm 8: FMG-Cycle Iteration

procedure FMG_CYCLE($\boldsymbol{v}, \boldsymbol{s}, Y$, P_list)
  if P_list is not empty **then**
    $P \leftarrow$ P_list[0].
    $Y_C \leftarrow \overline{P^T \text{diag}(\boldsymbol{v})} Y \text{diag}(\boldsymbol{v}) P$.
    $\boldsymbol{s}_C \leftarrow P^T \boldsymbol{s}$.
    $\boldsymbol{v}_C \leftarrow$ FMG_CYCLE($\mathbf{1}, \boldsymbol{s}_C, Y_C$, P_list[1:]).
    $\boldsymbol{v} \leftarrow D_{\boldsymbol{v}} P \boldsymbol{v}_C$.
  end if
  $\boldsymbol{v} \leftarrow P$ V_CYCLE($\boldsymbol{v}, \boldsymbol{s}, Y$, P_list)
end procedure

for the classical Gauss-Seidel approach, showing that its local convergence relies on a sufficiently "Laplacian-like" admittance matrix (Approximation (3.14)) and power demands that are not too large. We will then leverage this to develop a local proof of convergence for the split Gauss-Seidel approach (Algorithm 3). Finally, we will use existing linear multigrid theory to analyze the rate of convergence of the two-grid version of Algorithm 7.

3.6.1 Linear Formulation Near the Solution

First, we establish the notation to express the power flow problem when a linear approximation is valid. Suppose $\boldsymbol{v}^*$ is the solution vector, and that our current iterate is

$$v_i = v_i^*(1 + m_i)e^{\iota\delta_i} \approx v_i(1 + m_i + \iota\delta_i), \quad m_i, \delta_i \in \mathbb{R}, \quad \forall\ i = 1, ..., n.$$

Also,

$$\overline{v_i}v_j = \overline{v_i^*}v_j^*(1 + m_i - \imath\delta_i + m_j + \imath\delta_j).$$

So,

$$\begin{aligned}\overline{v_i}\sum_{j=1}^{n} Y_{ij}v_j \approx &\overline{v_i^*}\sum_{j=1}^{n} Y_{ij}v_j^*\left(1 + m_i - \imath\delta_i + m_j + \imath\delta_j\right)\\ =&\overline{s_i}(1 + m_i - \imath\delta_i) + \overline{v_i^*}\sum_{j=1}^{n} Y_{ij}v_j^*(m_j + \imath\delta_j).\end{aligned}$$

Therefore, at the solution, we have for each i,

$$0 = s_i(m_i - \imath\delta_i) + \overline{v_i^*}\sum_{j=1}^{n} Y_{ij}v_j^*(m_j + \imath\delta_j). \tag{3.43}$$

That is, the solution occurs when we solve the system of equations (3.43). The solution of course is $m_i = \delta_i = 0$, but we must obtain this solution without being able to explicitly write down Equation (3.43), as we do not know v_i*.

We now write the system of equations to solve in terms of real numbers. Let rf denote the real formulation function; that is, if $c \in \mathbb{C}, \boldsymbol{x} \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n\times n}$, then

$$\mathrm{rf}(c) = \begin{bmatrix}\mathrm{Re}(c)\\ \mathrm{Im}(c)\end{bmatrix} \in \mathbb{R}^2 \qquad \mathrm{rf}(\boldsymbol{x}) = \begin{bmatrix}\mathrm{rf}(x_1)\\ \vdots\\ \mathrm{rf}(x_n)\end{bmatrix} \in \mathbb{R}^{2n}$$

$$\mathrm{rf}(A) = \begin{bmatrix}\mathrm{rf}(A_{11}) & \mathrm{rf}(\imath A_{11}) & \cdots & \mathrm{rf}(A_{1n}) & \mathrm{rf}(\imath A_{1n})\\ & & \ddots & & \\ \mathrm{rf}(A_{n1}) & \mathrm{rf}(\imath A_{n1}) & \cdots & \mathrm{rf}(A_{nn}) & \mathrm{rf}(\imath A_{nn})\end{bmatrix} \in \mathbb{R}^{2n\times 2n}.$$

Note that each 2-by-2 block of $\mathrm{rf}(A)$ is the 2-by-2 real-valued matrix representation of a complex number and that $\mathrm{rf}(A\boldsymbol{x}) = \mathrm{rf}(A)\,\mathrm{rf}(\boldsymbol{x})$.

We can write Equations (3.43) as

$$(A + B)\boldsymbol{x} = \boldsymbol{0}, \tag{3.44}$$

where

$$
\begin{aligned}
A &= \mathrm{rf}(\overline{D_v} Y D_v) \\
B &= \begin{bmatrix} \mathrm{rf}\,(\overline{s_1}) & \mathrm{rf}\,(-\imath\overline{s_1}) & & \\ & & \ddots & \\ & & & \mathrm{rf}\,(\overline{s_n}) \quad \mathrm{rf}\,(-\imath\overline{s_n}) \end{bmatrix} \\
\boldsymbol{x} &= \begin{bmatrix} m_1 & \delta_1 & \cdots & m_n & \delta_n \end{bmatrix}^T .
\end{aligned}
$$

Note that, unlike A, the 2-by-2 blocks of B are not real-valued matrix representations of complex numbers. When $\boldsymbol{v}$ is near the solution, finding the solution is approximately equivalent to solving the linear system of Equation (3.44).

We must modify the above slightly to account for PV nodes. At these nodes, the voltage magnitude is known, so $m_i = 0$ is constant. If i is a PV node, we make the replacements

$$
\begin{aligned}
\mathcal{A}_{ii} &= \begin{bmatrix} 1 & 0 \\ \mathrm{Re}(Y_{ii}|v_i^*|^2) & -\mathrm{Im}(Y_{ii}|v_i^*|^2) \end{bmatrix} \\
\mathcal{A}_{ij} &= \begin{bmatrix} 0 & 0 \\ \mathrm{Re}(Y_{ij}\overline{v_i^*}v_j^*) & -\mathrm{Im}(Y_{ij}\overline{v_i^*}v_j^*) \end{bmatrix} \\
\mathcal{B}_{ii} &= \begin{bmatrix} 0 & 0 \\ \mathrm{Re}(\overline{s_1}) & \mathrm{Im}(\overline{s_i}) \end{bmatrix}.
\end{aligned}
$$

In words, we moved the first row down to the second row and replaced the first row with all-zeros plus a 1 on the diagonal.

3.6.2 (A Slight Variant of) Classical Gauss-Seidel

Here we present a local convergence result of the classical Gauss-Seidel approach. For ease of analysis, we analyze a slight variant on the classical Gauss-Seidel approach: for each PV node, we iterate Equations (3.11)–(3.13) until convergence; this is equivalent to updating just the ith phase angle to match $\mathrm{Re}(s_i)$. Note that $m_i = 0$ is constant at PV nodes, so we could remove the row and column associated with those m_i without changing the algorithm. However, we find it more convenient in the analysis to keep all blocks of size 2-by-2.

We present a lemma, theorem, and corollary, each of which has conditions that are simpler but more restrictive than the last. The sufficient conditions given here for convergence are not met in any real-world test cases of which the authors are aware; however, there typically is a nearby network for which the conditions are met. So, the results still provide insight as to structural properties that tend to be good or bad for convergence. They show that the following properties are good for convergence:

- Large shunt admittances.
- An admittance matrix that is nearly a complex rotation of a real-valued matrix (Approximation (3.14)).
- Transmission line admittances adjacent to PV nodes that lie near the imaginary axis.

The following properties may be bad for convergence:

- Heavy loads on the network (i.e. large s).

- Neighboring nodes with large differences in voltage magnitudes or phase angles (often symptoms of a heavily loaded network).

The classical Gauss-Seidel approach is not exactly the same as applying a linear block Gauss-Seidel iteration with blocks of size 2. Instead, let $A = T + U$, where T is the 2-by-2 block lower triangular portion of A and U is the 2-by-2 block strictly upper triangular portion. Then the classical Gauss-Seidel approach can be expressed as the update

$$\boldsymbol{x} \leftarrow -T^{-1}(U + B)\boldsymbol{x}. \tag{3.45}$$

In the following, we denote as $\mathcal{A}_{ij}$ the $(i, j)th$ 2-by-2 block of the matrix A, and $\boldsymbol{\xi}_i$ the ith length-2 subvector of the vector $\boldsymbol{\xi}$.

We now prove conditions under which the classical Gauss-Seidel method converges. The convergence of the linear Gauss-Seidel method for strictly diagonally dominant linear systems is well known (see, e.g., Theorem 4.9 of [58]). A block diagonally dominant matrix is a matrix such that [27]:

$$|\lambda_{\min}(\mathcal{A}_{ii})| \geq \sum_{j \neq i} |\lambda_{\max}(\mathcal{A}_{ij})|.$$

A straightforward generalization of the proof for the Gauss-Seidel method on diagonally dominant matrices shows that linear block Gauss-Seidel converges on block diagonally dominant matrices. We construct the following proof similarly.

Recall that Y_{is} is the shunt admittance at node i.

Lemma 2. *Let*

$$m = \min_i |v_i^*| \qquad M = \max_i |v_i^*| \qquad \Delta = \max_{i,j\ nbrs} |\theta_i - \theta_j|.$$

Suppose that, for each node i that is PQ node,

$$m|Y_{ii}| > \frac{|s_i|}{m} + M \sum_{j \neq i} |Y_{ij}|, \tag{3.46}$$

and for each node i that is a PV node,

$$m|\operatorname{Im}(Y_{ii})| > \frac{|\operatorname{Im}(s_i)|}{m} + M \sum_{j \neq i} \Big(|\operatorname{Im}(Y_{ij})| + \Delta |\operatorname{Re}(Y_{ij})| \Big), \tag{3.47}$$

Then, if the current iterate is close enough that a linear approximation is valid, the classical Gauss-Seidel method converges.

Proof. From Equation (3.45), we see that the classical Gauss-Seidel method converges if and only if the eigenvalues of the matrix $T^{-1}(U + B)$ have magnitude less than 1.

Let $(\boldsymbol{\xi}, \mu)$ be an eigenvector, eigenvalue pair of the iteration matrix in Equation (3.45). Let i denote the length-2 block of $\boldsymbol{\xi}$ with the largest magnitude, and scale $\boldsymbol{\xi}$ so that $\|\boldsymbol{\xi}_i\| = 1$. First, suppose i is a PQ node. Then

$$\begin{aligned} \mu \left(-\sum_{j \leq i} \mathcal{A}_{ij} \boldsymbol{\xi}_j \right) &= \mathcal{B}_{ii} \boldsymbol{\xi}_i + \sum_{j > i} \mathcal{A}_{ij} \boldsymbol{\xi}_j \\ \implies |\mu| = \frac{\left\| \mathcal{B}_{ii} \boldsymbol{\xi}_i + \sum_{j>i} \mathcal{A}_{ij} \boldsymbol{\xi}_j \right\|}{\left\| \sum_{j \leq i} \mathcal{A}_{ij} \boldsymbol{\xi}_j \right\|} &\leq \frac{|\lambda_{\max}(B_{ii})| + \sum_{j>i} |\lambda_{\max}(A_{ij})|}{|\lambda_{\min}(A_{ii})| - \sum_{j<i} |\lambda_{\max}(A_{ij})|}. \end{aligned} \tag{3.48}$$

Now, all 2-by-2 blocks of A are 2-by-2 real-valued matrix representations of complex numbers, which have two eigenvalues, each of which have magnitude equal to the magnitude of the associated complex number. Therefore,

$$|\lambda_{\max}(\mathcal{A}_{ij})| = |\lambda_{\min}(\mathcal{A}_{ij})| = |Y_{ij} \overline{v_i^*} v_j^*|.$$

The block $\mathcal{B}_{ii}$ is of the form $\begin{bmatrix} a & b \\ b & -a \end{bmatrix}$, which is not a representation of a complex number. However, it is straightforward to confirm that such a matrix has eigen-

vectors

$$\begin{bmatrix} b \\ \sqrt{a^2+b^2}-a \end{bmatrix}, \begin{bmatrix} b \\ -\sqrt{a^2+b^2}-a \end{bmatrix}$$

with associated eignevalues $\sqrt{a^2+b^2}$ and $-\sqrt{a^2+b^2}$, respetively. Therefore,

$$|\lambda_{\max}(\mathcal{B}_{ij})| = |\lambda_{\min}(\mathcal{B}_{ij})| = |\overline{s_i}|.$$

So, we may rewrite Inequality (3.48) as

$$\begin{aligned} |\mu| \leq & \frac{|\overline{s_i}| + \sum_{j>i} |Y_{ij}\overline{v_i^*}v_j^*|}{|Y_{ii}\overline{v_i^*}v_i^*| - \sum_{j<i} |Y_{ij}\overline{v_i^*}v_j^*|} \leq \frac{|\overline{s_i}| + M|\overline{v_i^*}| \sum_{j>i} |Y_{ij}|}{m|\overline{v_i^*}||Y_{ii}| - M|\overline{v_i^*}| \sum_{j<i} |Y_{ij}|} \\ = & \frac{|\overline{s_i/v_i^*}| + M \sum_{j>i} |Y_{ij}|}{m|Y_{ii}| - M \sum_{j<i} |Y_{ij}|} = 1 + \frac{-m|Y_{ii}| + |\overline{s_i}/m| + M \sum_{j\neq i} |Y_{ij}|}{m|Y_{ii}| - M \sum_{j<i} |Y_{ij}|}. \end{aligned}$$

The second term in the above expression is strictly less than zero, by Assumption (3.46). Therefore, $|\mu| < 1$.

Now suppose that i is a PV node. The matrix $\mathcal{B}_{ii}$ has eigenvalues 0 and $\operatorname{Im}(\overline{s_i})$. Similarly, $\mathcal{A}_{ij}$ has eigenvalues 0 and $-\operatorname{Im}(Y_{ij}\overline{v_i^* v_j^*})$. The matrix $\mathcal{A}_{ii}$ has eigenvalues 1 and $-|v_i^*|^2 \operatorname{Im}(Y_{ii})$. We may scale Y and $\boldsymbol{s}$ by an arbitrary positive constant without changing the solution, so we may assume without loss of generality that $|v_i^*|^2 |\operatorname{Im}(Y_{ii})| < 1$. So,

$$\begin{aligned} |\lambda_{\max}(\mathcal{B}_{ij})| = & |\operatorname{Im}(\overline{s_i})| \\ |\lambda_{\max}(\mathcal{A}_{ij})| = & |\operatorname{Im}(Y_{ij}\overline{v_i^*}v_j^*)| \\ |\lambda_{\min}(\mathcal{A}_{ii})| = & |v_i^*|^2 |\operatorname{Im}(Y_{ii})|. \end{aligned}$$

What follows is similar to the PQ case:

$$\begin{aligned}
\mu\left(\sum_{j\le i}\mathcal{A}_{ij}\boldsymbol{\xi}_j\right) &= \mathcal{B}_{ii}\boldsymbol{\xi}_i + \sum_{j>i}\mathcal{A}_{ij}\boldsymbol{\xi}_j \\
\implies |\mu| = \frac{\left\|\mathcal{B}_{ii}\boldsymbol{\xi}_i + \sum_{j>i}\mathcal{A}_{ij}\boldsymbol{\xi}_j\right\|}{\left\|\sum_{j\le i}\mathcal{A}_{ij}\boldsymbol{\xi}_j\right\|} &\le \frac{|\lambda_{\max}(\mathcal{B}_{ii})| + \sum_{j>i}|\lambda_{\max}(\mathcal{A}_{ij})|}{|\lambda_{\min}(\mathcal{A}_{ii})| - \sum_{j<i}|\lambda_{\max}(\mathcal{A}_{ij})|} \\
&\le \frac{|\operatorname{Im}(s_i)|/m + M\sum_{j>i}|\operatorname{Im}(Y_{ij}e^{\imath(\theta_j-\theta_i)})|}{m|\operatorname{Im}(Y_{ii})| - M\sum_{j<i}|\operatorname{Im}(Y_{ij}e^{\imath(\theta_j-\theta_i)})|} \\
&\le \frac{|\operatorname{Im}(s_i)|/m + M\sum_{j>i}|\operatorname{Im}(Y_{ij})| + \Delta|\operatorname{Re}(Y_{ij})|}{m|\operatorname{Im}(Y_{ii})| - M\sum_{j<i}|\operatorname{Im}(Y_{ij})| + \Delta|\operatorname{Re}(Y_{ij})|}.
\end{aligned}$$

By Assumption (3.47), this is less than 1. □

Theorem 3. *Suppose $m \le 1.0 \le M$, and let $\widetilde{M} = \max(M, 1/m)$. Suppose that Approximation* (3.14) *is exact, and suppose that*

$$|Y_{is}| > \widetilde{M}^2\frac{|s_i|}{|\operatorname{Im}(z)|} + \left(\widetilde{M}^2 - 1\right)\left(1 + \Delta\left|\frac{\operatorname{Re}(z)}{\operatorname{Im}(z)}\right|\right)\sum_{j\ne i}|Y_{ij}|. \tag{3.49}$$

Then, if the current iterate is close enough that a linear approximation is valid, the classical Gauss-Seidel method converges.

Proof. First, for PQ nodes, note that Condition (3.49) implies

$$|Y_{is}| > \widetilde{M}^2|s_i| + \left(\widetilde{M}^2 - 1\right)\sum_{j\ne i}|Y_{ij}|.$$

Because Approximation (3.14) is exact,

$$|Y_{ii}| = |Y_{is}| + \sum_{j\ne i}|Y_{ij}|.$$

So, we can rewrite Condition (3.49) as

$$|Y_{ii}| > \widetilde{M}^2|s_i| + \widetilde{M}^2\sum_{j\ne i}|Y_{ij}| > \frac{|s_i|}{m^2} + \frac{M}{m}\sum_{j\ne i}|Y_{ij}|,$$

which implies Condition (3.46) of Lemma 2.

For PV nodes, note that, because Approximation (3.14) is exact, $Y_{ij} = |Y_{ij}|(\mathrm{Re}(z) + \imath\,\mathrm{Im}(z))$. Condition (3.49) implies that

$$|\,\mathrm{Im}(z)||Y_{is}| > \frac{|\overline{s_i}|}{m^2} + (M/m - 1)\sum_{j\neq i}|Y_{ij}|(|\,\mathrm{Im}(z)| + \Delta|\,\mathrm{Re}(z)|)$$

$$\implies m|\,\mathrm{Im}(Y_{ii})| > \frac{|\overline{s_i}|}{m} + M\sum_{j\neq i}(|\,\mathrm{Im}(Y_{ij})| + \Delta|\,\mathrm{Re}(Y_{ij})|).$$

Condition (3.47) of Lemma 2 is therefore met at PV nodes. So, by Theorem 2, the classical Gauss-Seidel method converges. □

The following corollary is valid under the DC Load Flow Approximation of Section 3.3.1. We omit the proof.

Corollary 1. *Suppose that $m = M = 1.0$, that Approximation* (3.14) *is exact, and that $z = \imath$. Then*

$$|Y_{is}| > |s_i| \quad \forall\ i \tag{3.50}$$

implies convergence of the classical Gauss-Seidel method if a linear approximation is valid.

3.6.3 Split Gauss-Seidel

We now analyze the one-level split Gauss-Seidel method discussed in Section 3.4.2. In this section, we need not modify the algorithm for PV nodes as in Section 3.6.2, as our approach solves for the phase angle directly. The results in this section show that convergence of the split Gauss-Seidel approach is quite similar to that of the classical Gauss-Seidel approach when Approximation (3.14) is appropriate.

Here we again write the local system as in Equation (3.44), but here rotate the equations by $\bar{z}$ before splitting them into real and imaginary components. Prior to the inclusion of PV nodes, this results in

$$A = \mathrm{rf}\left(\bar{z}\overline{D_v Y D_v}\right) \tag{3.51}$$

$$B = \begin{bmatrix} \mathrm{rf}(\overline{zs_1}) & \mathrm{rf}(-\overline{\iota z s_1}) & & \\ & & \ddots & \\ & & & \mathrm{rf}(\overline{zs_n}) \quad \mathrm{rf}(-\overline{\iota z s_n}) \end{bmatrix} \tag{3.52}$$

If i is a PV node, rather than setting to zero the real component of Equation (3.43), the split Gauss-Seidel method sets to zero

$$0 = \mathrm{Im}\left[\bar{z}\left(s_i(m_i - \iota\delta_i) + \overline{v_i^*}\sum_{j=1}^{n} Y_{ij}v_j^*(m_j + \iota\delta_j)\right)\right]. \tag{3.53}$$

However, a PV update begins by updating s_i through Equation (3.11), and so the current iterates already matches $\mathrm{Im}(s_i)$ exactly. Thus, at a PV node, only the real component of Equation (3.43) is nonzero.

Therefore, if i is a PV node, we make the replacements

$$\mathcal{A}_{ii} = \begin{bmatrix} 1 & 0 \\ \mathrm{Re}(Y_{ii}|v_i^*|^2) & -\mathrm{Im}(Y_{ii}|v_i^*|^2) \end{bmatrix}$$

$$\mathcal{A}_{ij} = \begin{bmatrix} 0 & 0 \\ \mathrm{Re}(Y_{ij}\overline{v_i^*}v_j^*) & -\mathrm{Im}(Y_{ij}\overline{v_i^*}v_j^*) \end{bmatrix}$$

$$\mathcal{B}_{ii} = \begin{bmatrix} 0 & 0 \\ \mathrm{Re}(\overline{s_1}) & \mathrm{Im}(\overline{s_i}) \end{bmatrix}.$$

Note that this is valid assuming $\mathrm{Im}(z) \neq 0$.

We can describe the split Gauss-Seidel approach with the matrix splitting $A = \widetilde{T} + \widetilde{U}$, where $\widetilde{T}$ is the lower triangular portion of A (rather than the block

lower triangular portion as in L), and $\widetilde{U}$ is the strictly upper triangular portion of A.

Lemma 3. *Let*

$$m = \min_i |v_i^*| \qquad M = \max_i |v_i^*| \qquad \Delta = \max_{i,j \; nbrs} |\theta_i - \theta_j|.$$

Suppose that, for each node i that is a PQ node,

$$m|Y_{ii}| > \frac{|s_i| + 2M|\operatorname{Im}(\overline{z}Y_{ii})|}{m} + M\sum_{j\neq i}|Y_{ij}|, \tag{3.54}$$

and for each node i that is a PV node,

$$m|\operatorname{Im}(Y_{ii})| > \frac{|\operatorname{Im}(s_i)|}{m} + M\sum_{j\neq i}\left(|\operatorname{Im}(Y_{ij})| + \Delta|\operatorname{Re}(Y_{ij})|\right), \tag{3.55}$$

Then, if the current iterate is close enough that a linear approximation is valid, the split Gauss-Seidel method converges.

Proof. If i is a PV node, then the block $\mathcal{A}_{ii}$ is lower triangular, so $\mathcal{T}_{ii} = \widetilde{\mathcal{T}}_{ii}$. Therefore, the PV node case is the same as in Lemma 2.

For PQ nodes, we have

$$\begin{aligned}\mu\left(-\sum_{j\leq i}\widetilde{\mathcal{T}}_{ij}\boldsymbol{\xi}_j\right) &= \mu\left(\sum_{j\leq i}\widetilde{\mathcal{A}}_{ij} - W_i\boldsymbol{\xi}_i\right)\\ =\mathcal{B}_{ii}\boldsymbol{\xi}_i + \sum_{j\geq i}\mathcal{U}_{ij}\boldsymbol{\xi}_j &= \mathcal{B}_{ii}\boldsymbol{\xi}_i + W_i\boldsymbol{\xi}_i + \sum_{j>i}\mathcal{A}_{ij}\boldsymbol{\xi}_j,\end{aligned}$$

where

$$W_i = \begin{bmatrix} 0 & -|v_i^*|^2\operatorname{Im}(\overline{z}Y_{ii}) \\ 0 & 0 \end{bmatrix}.$$

The matrix T_i has two eigenvalues, 0 and $-|v_i^*|^2\operatorname{Im}(\overline{z}Y_{ii})$, and all other matrices have eigenvalues as described in the proof of Lemma 2. Therefore,

$$|\mu| \leq \frac{|\overline{s_i}| + |v_i^*|^2|\operatorname{Im}(\overline{z}Y_{ii})| + \sum_{j>i}|Y_{ij}\overline{v_i^*}v_j^*|}{|Y_{ii}\overline{v_i^*}v_i^*| - |v_i^*|^2|\operatorname{Im}(\overline{z}Y_{ii})| - \sum_{j<i}|Y_{ij}\overline{v_i^*}v_j^*|} \leq \frac{|\overline{s_i/m}| + M|\operatorname{Im}(\overline{z}Y_{ii})| + M\sum_{j>i}|Y_{ij}|}{m|Y_{ii}| - M|\operatorname{Im}(\overline{z}Y_{ii})| - M\sum_{j<i}|Y_{ij}|} < 1,$$

by Condition (3.54). □

Note that Condition (3.54) is the same as Condition (3.46) with a (typically small) extra term, and Condition (3.55) is exactly the same as Condition (3.47). Thus, the classical and split Gauss-Seidel methods have similar convergence properties.

Theorem 4. *Suppose $m \leq 1.0 \leq M$, and let $\widetilde{M} = \max(M, 1/m)$. Suppose that Approximation* (3.14) *is exact, and suppose that*

$$|Y_{is}| > \widetilde{M}^2 \frac{|s_i|}{|\operatorname{Im}(z)|} + \left(\widetilde{M}^2 - 1\right)\left(1 + \Delta\left|\frac{\operatorname{Re}(z)}{\operatorname{Im}(z)}\right|\right)\sum_{j \neq i} |Y_{ij}|. \tag{3.56}$$

Then, if the current iterate is close enough that a linear approximation is valid, the split Gauss-Seidel method converges.

Proof. If Approximation (3.14) is exact, then $\operatorname{Im}(\overline{z}Y_{ii}) = 0$, so $T = \widetilde{T}$ and $U = \widetilde{U}$. Therefore, the split and classical Gauss-Seidel methods have the same convergence properties, and so Theorem 3 applies. □

Again, we omit the proof of the following corollary.

Corollary 2. *Suppose that $m = M = 1.0$, that Approximation* (3.14) *is exact, and that $z = \iota$. Then*

$$|Y_{is}| > |s_i| \quad \forall\ i \tag{3.57}$$

implies convergence of the split Gauss-Seidel method if a linear approximation is valid.

3.6.4 Multigrid Power Flow

In this section we analyze the multigrid power flow method described in Section 3.5. We focus on the multigrid iteration with a Gauss-Seidel smoother and a full solve with Newton's method at the second hierarchy level.

As shown in Theorem (2), an update on the coarse problem is equivalent to a group-synchronous update on the original problem. In this case near the solution in which a linear approximation is valid, this implies a group-synchronous update on the linear system (3.44). But in Section 3.3.2 we showed that group-synchronous updates on linear systems are equivalent to updates on coarse linear systems. Therefore, if a linear approximation to the power flow problem is valid, the multigrid method presented in this paper is equivalent to a linear multigrid method.

Lemma 4. *Let C denote a symmetric and positive-definite real-valued matrix, and let C' denote the matrix with the ith row and column of C removed. Then C' is symmetric and positive-definite.*

Proof. It is clear that C' is symmetric. Suppose that $\boldsymbol{x}$ is a vector such that $\boldsymbol{x}^T C' \boldsymbol{x} < 0$. Let

$$\boldsymbol{y} = [\boldsymbol{x}_{:i-1}\ 0\ \boldsymbol{x}_{i:}]^T.$$

Then $\boldsymbol{y}^T C \boldsymbol{y} = \boldsymbol{x}^T C' \boldsymbol{x} < 0$, and so C is not positive-definite, a contradiction. □

Lemma 5. *Let $A' + B'$ denote the system matrix with rows and columns associated with the magnitude change m_i of PV nodes removed. Suppose that Approximation* (3.14) *is exact and that*

$$|Y_{is}| > |s_i| \quad \forall\ i.$$

Additionally, suppose that either there are no PV nodes in the network, or that $z = \imath$. Then the reduced system matrix $A' + B'$ is symmetric and positive-definite.

Proof. First, suppose there are no PV nodes in the network. Split Y into Y_1 and Y_2, where Y_2 is the part due to shunt admittances, and Y_1 is everything else. Then

we have, for appropriate L_1 and L_2,

$$\begin{aligned} Y_1 &= zL_1 \\ Y_2 &= zL_2 \\ A_1 &= \mathrm{rf}\left(\overline{D_v L_1 D_v}\right) \\ A_2 &= \mathrm{rf}\left(\overline{D_v L_2 D_v}\right). \end{aligned}$$

The matrix L_1 is a graph Laplacian, so $\overline{D_v L_1 D_v}$ is Hermitian positive-definite, so A_1 is symmetric positive-definite. The matrix L_2 is a positive-definite diagonal matrix, so $\overline{D_v L_2 D_v}$ is also a positive-definite diagonal matrix, so A_2 is as well.

Each 2-by-2 diagonal block of A_2 is a diagonal matrix with entries $|Y_{is}|$. The matrix $\mathcal{B}_{ii}$ has eigenvalues $\pm|s_i|$, as shown in the proof of Lemma 2. Because $|Y_{is}| > |s_i|$ by assumption, we therefore have that $A_2 + B$ is symmetric and positive-definite, so $A + B$ is symmetric and positive-definite.

Now, to account for PV nodes, we assume that $z = \imath$. Then Equation (3.53) and the real component of Equation (3.43) are identical. So, consider A and B as described in Equations (3.51)–(3.52), before having made changes for block rows of PV nodes. As argued above this matrix is symmetric and positive-definite. Now simply remove the rows and columns of m_i, where i is a PV node, to obtain $A' + B'$. By Lemma 4, $A' + B'$ is symmetric and positive-definite. □

As discussed at the beginning of Section 3.6.2, $m_i = 0$ is kept constant at all times at PV nodes, so removing these rows and columns does not change the description of the algorithm.

In [25], Falgout et al. develop an estimate for the convergence rate of two-grid methods on symmetric positive-definite linear systems. Let $A' = \widetilde{T}' + \widetilde{U}'$

denote the splitting of A used in Section 3.6.3 with the magnitude-associated rows and columns removed as in A'. Then Falgout et al. show that the error iteration matrix for the two-grid method is

$$E_{TG} = (I - \widetilde{T}'^{-T}A)(I - PA_c^{-1}P^TA)(I - \widetilde{T}'^{-1}A), \tag{3.58}$$

where $A_c = P^TAP$. Further, let

$$\begin{aligned}
\widetilde{M} =& \widetilde{T}'^T\left(\widetilde{T}'^T + \widetilde{T}' - A\right)^{-1}\widetilde{T}' = \widetilde{T}'^T D^{-1}_{\text{diag}(A)}\widetilde{T}' \\
\pi_A =& PA_c^{-1}P^TA \\
\widetilde{M}_c =& P^T\widetilde{M}P \\
\pi_{\widetilde{M}} =& P\widetilde{M}_c^{-1}P^T\widetilde{M}.
\end{aligned}$$

Then Theorem 4.3 of [25] shows that the convergence rate of the two-grid method is $\rho(E_{TG}) = 1 - K_{TG}^{-1}$, where

$$K_{TG} = \sup_{\boldsymbol{v}} \frac{((I - \pi_{\widetilde{M}})\boldsymbol{v})^T\,\widetilde{M}(I - \pi_{\widetilde{M}})\boldsymbol{v}}{((I - \pi_A)\boldsymbol{v})^T\,A(I - \pi_A)\boldsymbol{v}} = \sup_{\boldsymbol{v}} \frac{\boldsymbol{v}^T\widetilde{M}(I - \pi_{\widetilde{M}})\boldsymbol{v}}{\boldsymbol{v}^TA\boldsymbol{v}}. \tag{3.59}$$

It is a fact that $(I - \widetilde{M}^{-1}A) = (I - \widetilde{T}'^{-1}A)(I - \widetilde{T}'^{-T}A)$, so $\widetilde{M}$ is simply a symmetrized vesion of $\widetilde{T}'$.

Power grids are built largely on the surface of the Earth, and so they tend to have a grid-like structure, with some variation due to overlapping lines. Therefore, the same experience in the multigrid community that multigrid methods are usually effective on graph-structured matrices with grid-like graphs applies here. We can compare this to the one-grid split Gauss-Seidel method by noting that that corresponds to setting $P = 0$, and so

$$K_{OL} = \sup_{\boldsymbol{v}} \frac{\boldsymbol{v}^T\widetilde{M}\boldsymbol{v}}{\boldsymbol{v}^TA\boldsymbol{v}}.$$

If $\boldsymbol{v}$ varies smoothly over the surface of the grid, then K_{OL} will be large, while $\pi_{\widetilde{M}}\boldsymbol{v} \approx \boldsymbol{v}$, and so K_{TG} is much smaller.

3.7 Convergence Experiments

In this section we demonstrate the convergence properties of multigrid power flow. We explore several variants of multigrid power flow by changing the method in the following ways:

- Whether to use Gauss-Seidel iterations or Jacobi iterations as the smoother.
- Whether to use V-Cycles or Full Multigrid Cycles.
- Whether to use the complete multigrid hierarchy or to switch to Newton's method at some level h. This case is further subdivided into the level at which to switch to Newton's method, and whether to use a full Newton or quasi-Newton method in the inner iteration. In this paper, we consider full Newton's method, but consider not running the inner iteration to convergence.

3.7.1 Real-World Networks

In this section we experiment with various real-world test networks under a variety of algorithm choices. We use here four real-world networks: the IEEE 57-bus, 118-bus, and 300-bus test networks, as well as the 1999-2000 Polish Winter Peak network (2,383 nodes) [82]. We run each test from a starting vector of $\mathbf{1}$, and continue until the relative residual drops below 10^{-6}. The Jacobi weighting schedules for the different networks were found by trial and error, and can be seen in Table 3.1. Note that we do not use the small-angle update approximation discussed at the end of Section 3.4.2.

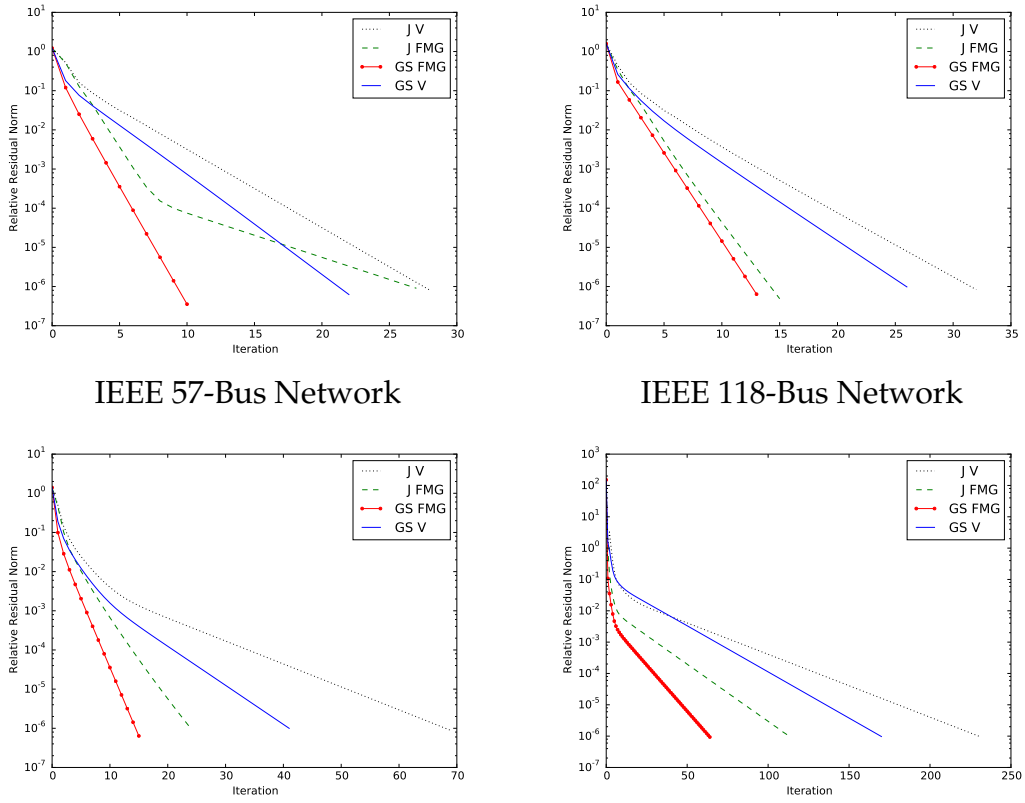
$\|\boldsymbol{r}\|_\infty$ in range	IEEE 57	IEEE 118	IEEE 300	Polish Winter Peak
$(0, 0.05]$	0.8	0.8	0.55	0.65
$(0.05, 0.1]$	0.75	0.75	0.55	0.65
$(0.1, 1.0]$	0.65	0.65	0.65	0.65
$(1.0, 5.0]$	0.3	0.3	0.3	0.3
$(5.0, 8.0]$	0.1	0.1	0.1	0.1
$(8.0, 50]$	0.05	0.05	0.05	0.05
$(50, 80]$	0.02	0.02	0.02	0.02
$(80, 200]$	0.01	0.01	0.01	0.01
$(200, \infty)$	0.01	0.01	0.001	0.01

Table 3.1: Jacobi weighting schedules for various real-world networks.

Although the real-world networks shown here have a variety of sizes, one should not interpret the results in this section as a rigorous scaling study. Each network has its own characteristics aside from size that can affect convergence. For example, the Polish Winter Peak network is a snapshot of the Polish power grid while it was under heavy load; the results of Section 3.6 show that this is likely to hinder convergence.

Figure 3.3 shows convergence plots for the four real-world test networks. We compare V-Cyles against FMG-Cycles and Gauss-Seidel smoothers against Jacobi smoothers. As expected, FMG-Cycles and Gauss-Seidel smoothers have stronger convergence properties. However, the V-Cycle is typically about half as expensive as the FMG-Cycle, and the Jacobi smoother is parallelizable while the Gauss-Seidel smoother is not. We note in these plots that many of the plots drop dramatically, and then "bend right" at some point as convergence slows somewhat. This is particular noticeable in the Polish Network: If a relative error of 10^{-3} is acceptable, then the FMG-Cycle with a Gauss-Seidel smoother is highly effective.

Next we show plot of iterations required to converge under a greater vari-



IEEE 57-Bus Network | IEEE 118-Bus Network

IEEE 300-Bus Network | Polish Network (2,383 nodes)

Figure 3.3: Convergence plots for various real-world test networks that compare V-Cycles against FMG-Cycles and Gauss-Seidel smoothers against Jacobi smoothers.

ety of algorithm choices. We consider the same cases as above, but here we also consider "cutting off" the traversal up the hierarchy at some point and applying a Newton solver. We investigate the effects of using a single Newton iterations, two iterations, and running the Newton solver to convergence (which we implemented as either convergence or 25 iterations, whichever came first).

Figures 3.4 and 3.5 show the results for the IEEE 57 and 118-bus test networks. Again, FMG-Cycles and Gauss-Seidel smoothers have stronger convergence. We see here that convergence rates are not drastically affected by using a

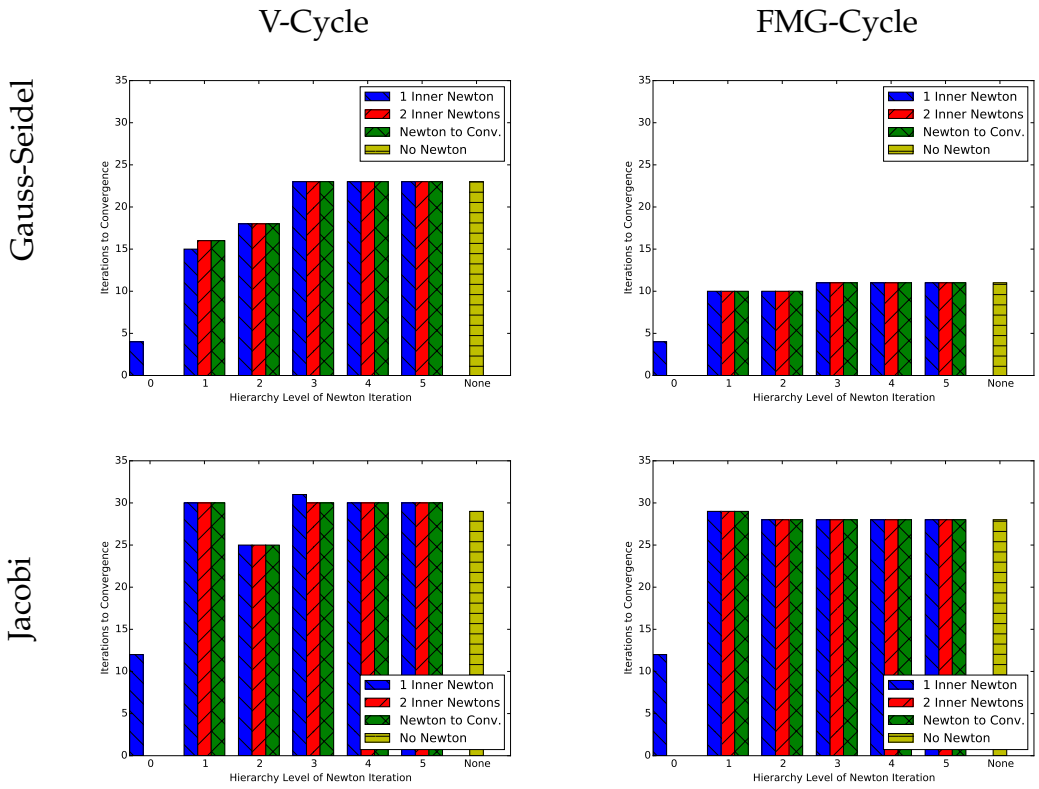


Figure 3.4: Multigrid power flow on the IEEE 57-bus test network with switches to Newton's method partway up the hierarchy. The bar group shows the hierarchy level at which we switch, while the bar within the group shows the number of inner Newton iterations performed. In blue is the test of the full hierarchy with no switch to Newton. Performing a single Newton iteration on hierarchy level 0 with $\omega = 1$ (as in the Gauss-Seidel row) is equivalent to a full Newton iteration.

Newton solver, with perhaps some moderate improvements if using a Newton solver at coarse level $h = 2$ (that is, a two-grid method). Running the Newton solver for more than a single iteration does not provide any benefit, which suggests that a quasi-Newton iteration might also be effective here. The Jacobi experiments show that the price of parallelism is worse convergence, but not drastically so.

We show the experiments with the IEEE 300-bus network in Figure 3.6. Here

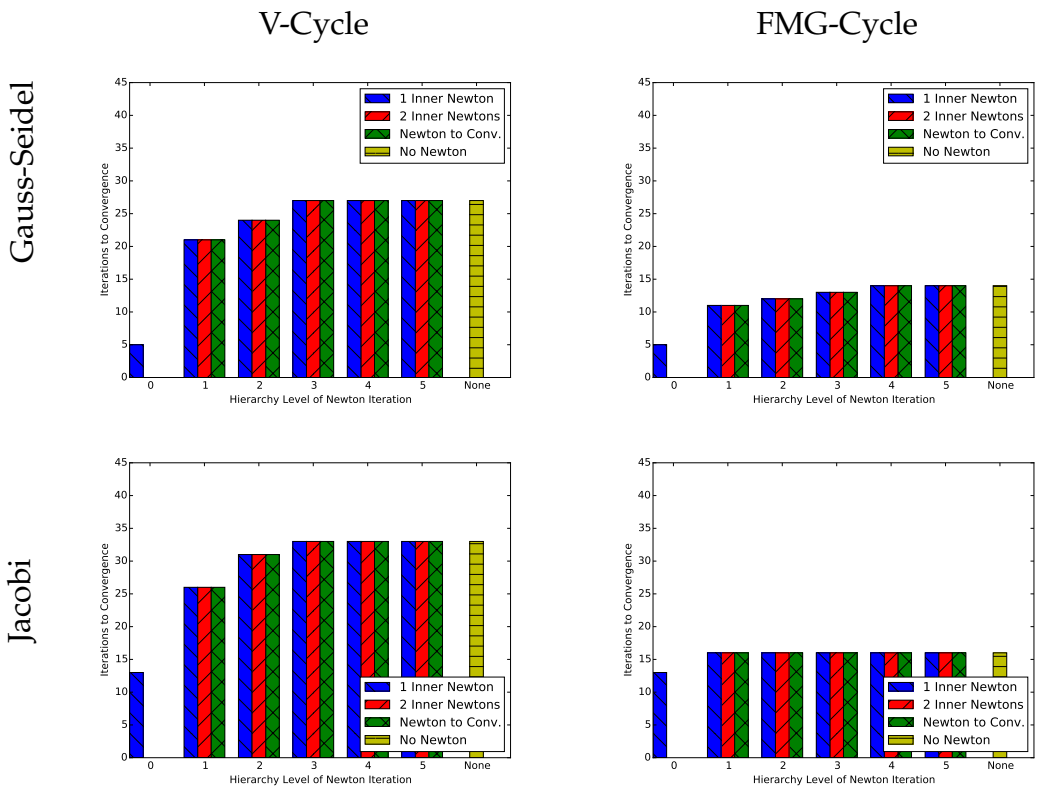


Figure 3.5: Multigrid power flow on the IEEE 118-bus test network with switches to Newton's method partway up the hierarchy.

the decision where and if to use a Newton solver has a much bigger effect on convergence rate when using a V-Cycle, especially for the Jacobi smoother. Again, we see that more than a single Newton iteration does not offer any improvement in rate of convergence.

The experiments with the 1999-2000 Polish Winter Peak network, shown in Figure 3.7, show still-greater impact of the Newton solver. Unlike in the IEEE 300-bus case, however, a Newton solver at coarse level $h = 2$ (that is, a two-grid method) helps the most with the Gauss-Seidel iteration, and in fact makes the Jacobi iteration worse.

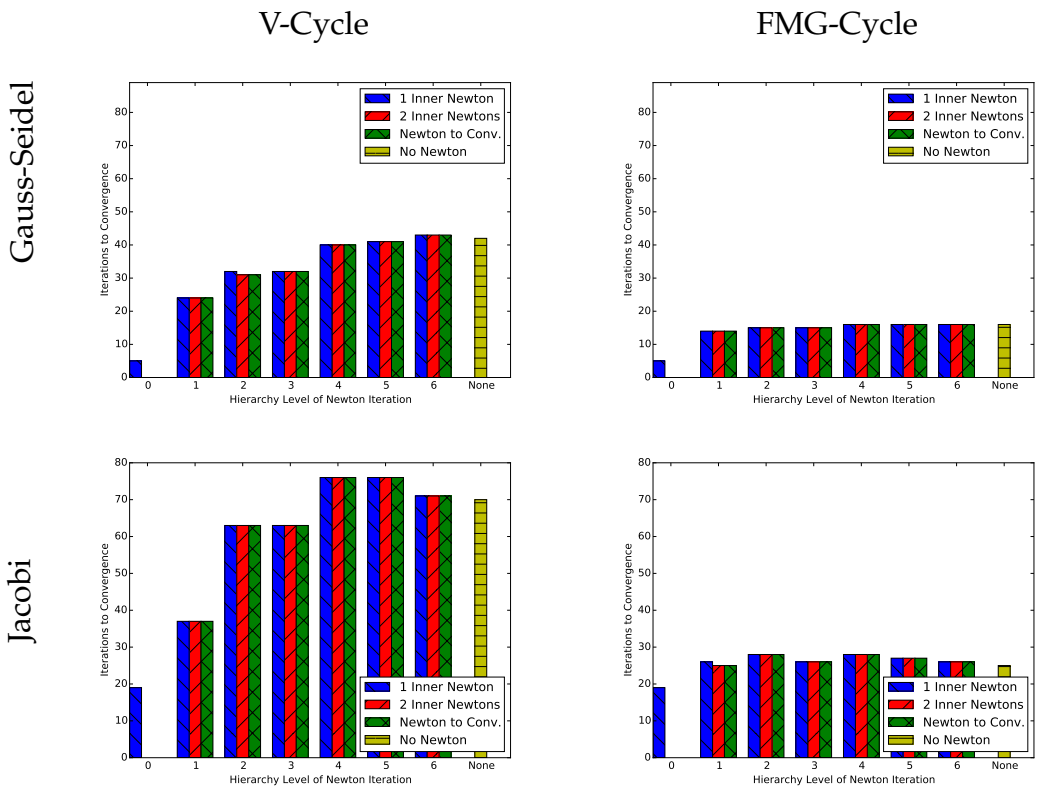


Figure 3.6: Multigrid power flow on the IEEE 300-bus test network with switches to Newton's method partway up the hierarchy.

3.7.2 Synthetic Networks

In this section we present scaling studies of multigrid power flow. We cannot simply scale up real-world problems, as those problems come from specific networks, and as discussed above the various real-world networks are not necessarily comparable. Instead, we generate synthetic networks in the following manner: we define some domain D and a smooth function $v(\boldsymbol{x})$ representing the voltage value at each point $\boldsymbol{x} \in D$. We can then place an arbitrary number of simulated nodes within D, and the function $v()$ gives us the voltage at each node. An application of Equation (3.1) then allows us to determine both the voltage and the power injection at each node. We then apply some means of

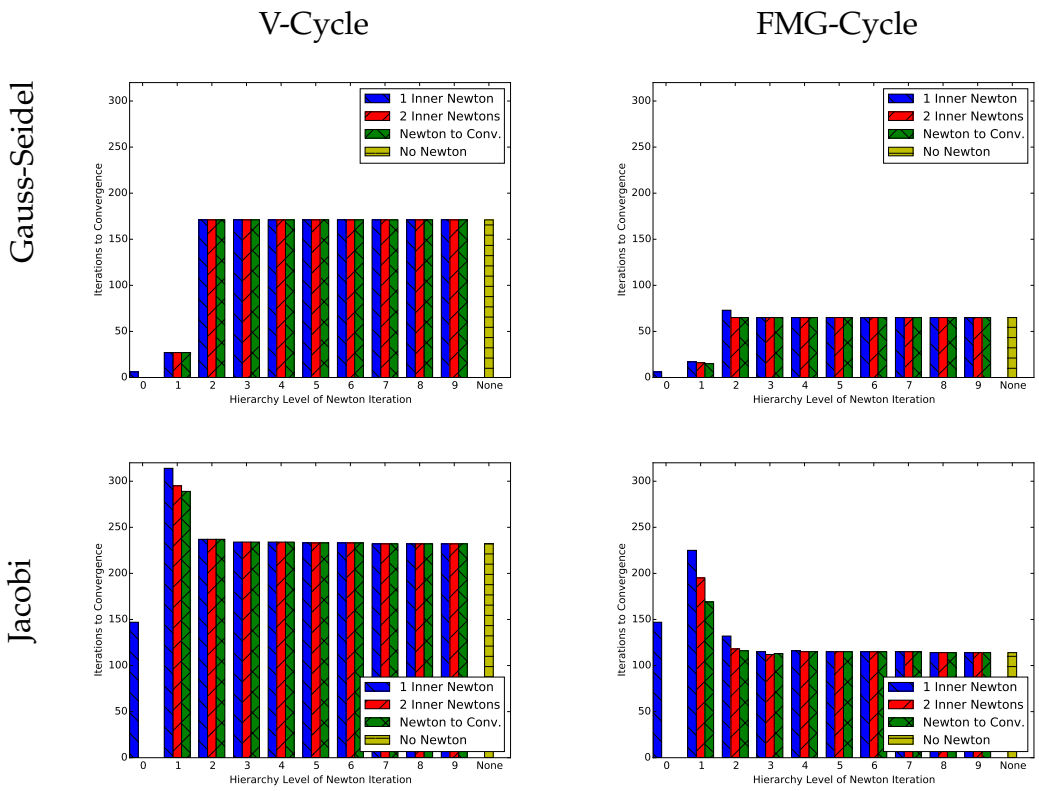


Figure 3.7: Multigrid power flow on the 1999-2000 Polish Winter Peak network with switches to Newton's method partway up the hierarchy.

determining the type of each node (PQ, PV, or slack) to complete our synthetic network construction. In this way, we can construct a sequence of comparable problems of varying sizes.

Here we define D to be a circle of radius 1 centered at the origin in the plane. The voltage magnitude and phase angle are defined separately as solutions to the Laplace equation:

$$\Delta u(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \in D$$

$$u(\boldsymbol{x}) = g \quad \boldsymbol{x} \in \partial D.$$

The solution to this problem is well known (see e.g. [23]) and is given by

$$u(\boldsymbol{x}) = u(r,\theta) = \int_{\partial D} \frac{1-r^2}{2\pi\|\boldsymbol{x}-\boldsymbol{y}\|_2^2} g(\boldsymbol{y}) dS(\boldsymbol{y}).$$

We use the solution to the Laplace equation because it is harmonic, which implies that it is smooth and also guarantees that all function values in D lie within the extrema of g on ∂D. In addition, we have the option of producing a variety of different smooth functions simply by changing the boundary values g.

For the voltage magnitude, we set

$$g(\boldsymbol{x}) = g(r,\theta) = 1 + m_M \cos(m_f \theta)$$

where m_M and m_f are chosen constant. Similarly, for the voltage phase angle we set

$$g(\boldsymbol{x}) = g(r,\theta) = a_M \cos(a_f \theta).$$

where a_M and a_f are chosen constants. Here we select

$$m_M = 0.05 \quad m_f = 1.0 \quad a_M = \pi/4 \quad a_f = 1.5.$$

Plots of these functions over the unit circle can be seen in Figure 3.8.

We choose nodes in a rectangular grid within the circle such that all neighboring nodes are a constant distance away. All lines have impedance $10 - \imath 20$ per unit. All lines are also given a line charging susceptance of 0.002 per unit. We place the slack node at the center of the circle, and all other nodes are randomly selected as PV nodes with probability 0.05, and PQ otherwise.

As above, we perform all tests with a flat start, where all elements of the starting voltage vector are $1 + \imath 0$, except for PV nodes, whose magnitudes are known, and the slack nodes, whose voltage is known. We then test the number of cycle iterations required for convergence. We perform the following 8 tests:

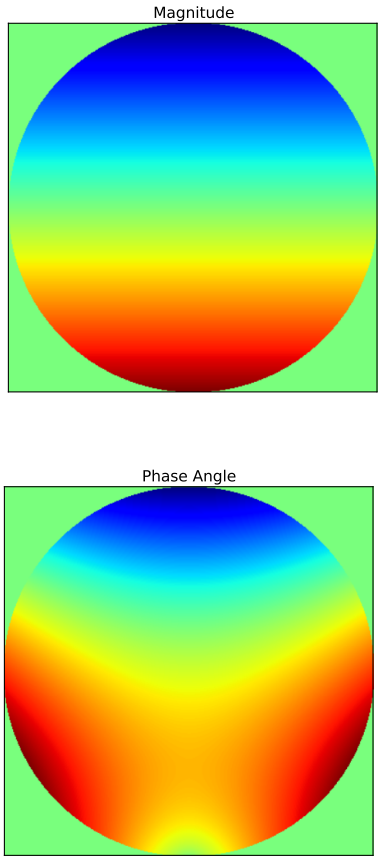


Figure 3.8: Analytic functions on the unit circle from which we sample voltage magnitude and phase angle to create synthetic networks.

- With Gauss-Seidel smoothing and V-cycles, no Newton iteration or 1 Newton step at level $h = 2$.
- With Gauss-Seidel smoothing and FMG-cycles, no Newton iteration or 1 Newton step at level $h = 2$.
- With Jacobi smoothing and V-cycles, no Newton iteration or 1 Newton step at level $h = 2$.
- With Jacobi smoothing and FMG-cycles, no Newton iteration or 1 Newton step at level $h = 2$.

When using a Jacobi smoother, we use the ω schedule that is used for the IEEE 57-bus and 118-bus networks in Table 3.1.

We show the results in Figure 3.9. As expected, the Gauss-Seidel smoother has better convergence than the Jacobi smoother. Above network size approximately 2^9, the FMG-Cycle shows high scalability, with iteration growth rate of approximately $O(n^{4/3})$. A single Newton iteration at coarse level $h = 2$ with Gauss-Seidel smoothing shows no decay in rate of convergence as network size grows, regardless of cycle type. It is also interesting that in most cases, convergence rate variance decreases with increasing network size.

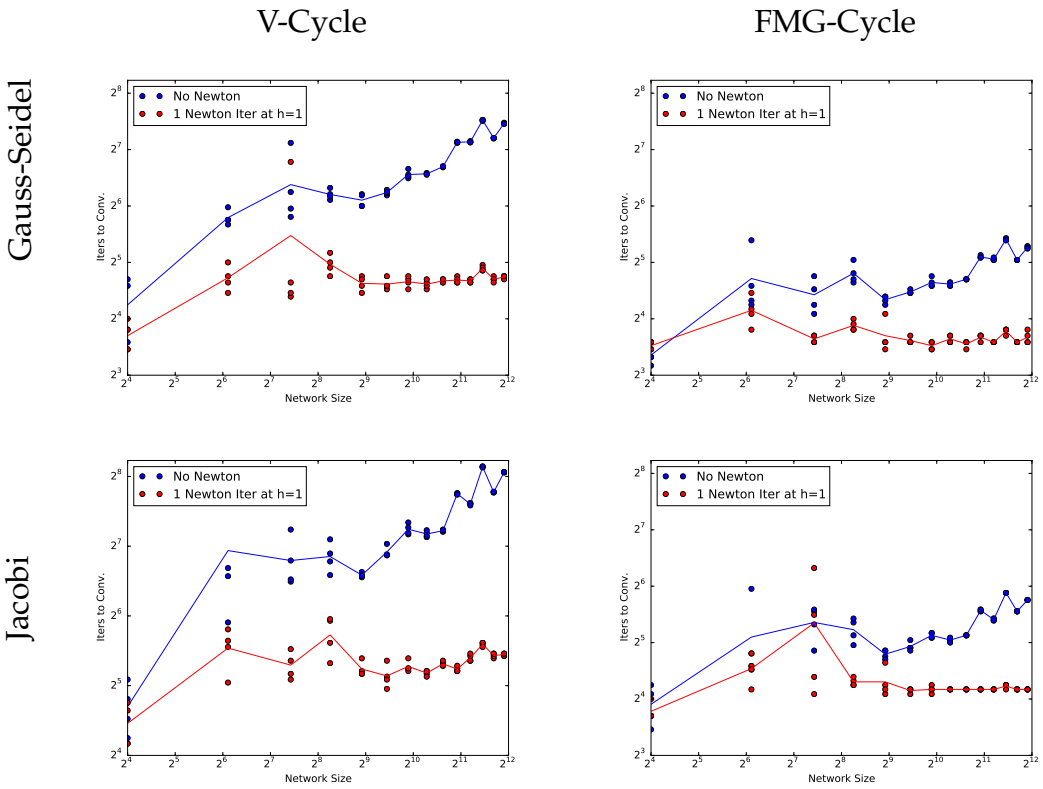


Figure 3.9: Convergence rates using a synthetic network. Each network size is sampled four times. The solid lines represent the mean at each network size. A single Newton iterations at coarse level $h = 2$ with Gauss-Seidel smoothing perform best, showing no decay in convergence as network size grows. A FMG-Cycle with either Jacobi or Gauss-Seidel smoothing also shows little decay in convergence rate.

CHAPTER 4

SOLVING GRAPH LAPLACIAN SYSTEMS THROUGH RECURSIVE BISECTIONS AND TWO-GRID PRECONDITIONING

Colin Ponce and Panayot Vassilevski

We present a parallelizable direct method for computing the solution to graph Laplacian-based linear systems derived from graphs that can be hierarchically bipartitioned with small edge cuts. For a graph of size n with constant-size edge cuts, our method decomposes a graph Laplacian in time $O(n \log n)$, and then uses that decomposition to perform a linear solve in time $O(n \log n)$.

We then use the developed technique to design a preconditioner for graph Laplacians that do not have this property. Finally, we augment this preconditioner with a two-grid method that accounts for much of the preconditioner's weaknesses. We present an analysis of this method, as well as a general theorem for the condition number of a general class of two-grid support graph-based preconditioners. Numerical experiments illustrate the performance of the studied methods.

This paper has been submitted for publication to SIAM Journal on Matrix Analysis and Applications.

4.1 Introduction

Recent years have seen an explosion of interest in studying large networks. The graphs representing these networks often reach hundreds of millions or billions of nodes; as a result, only the most scalable of algorithms are practical in network analysis.

One problem often of interest is solving linear systems based on graph Laplacians. Graph Laplacians appear in many areas, such as information dispersal through social networks or electricity flow through resistor networks. When graphs are large, general sparse linear solvers are not typically fast enough to be feasible. Thus, we must develop specialized graph Laplacian linear solvers.

In this paper we first consider graphs that can be hierarchically bipartitioned with small edge cuts. This graph structure appears in networks in which a small number of "high-bandwidth" interconnects stretch between distant clusters of nodes. For example, transportation systems and wide area computer networks often have this property.

Our method takes a two-step approach: first, perform a fast computation that results in highly structured error; second, exploit that structure to make error correction cheap. In our case, we recursively solve linear systems over each of the isolated graph partitions, ignoring the edge cuts. This results in an error vector that lies in a low-dimensional subspace determined by the edge cut. We then correct the error using the Sherman-Morrison-Woodbury formula.

We then extend this method to develop a preconditioner for graph Lapla-

cians that lack this property. We recursively partition the graph and remove between-partition edges as needed until we obtain a *support graph* that has the desired property. The solution of linear systems of the graph Laplacian of this support graph acts as our preconditioner.

Finally, we accelerate this preconditioner using two-grid techniques. A simple node aggregation technique leads to a coarse basis whose span often approximately captures the error modes that most damage the condition number of the preconditioned system. We develop a formula for the condition number of two-grid support graph preconditioned systems, and use this to develop guidelines for support graph creation.

Our direct solver is similar in spirit to nested dissection [29, 43]. This technique is also a direct solution method for linear systems based on graphs. It recursively splits the graph into two roughly equal sets by finding a separating set of nodes. Ordering the system matrix according to this recursive partitioning leads to a fast method for such linear systems. While our method also uses a recursive partitioning of a graph, it is based on edge separators instead of node separators.

Support graph preconditioning has grown in popularity since Spielman and Teng's seminal work on using ultra-sparsifiers to create support graphs with good conditioning [63]. Since its publication, various authors have improved on the graph-theoretic algorithms underlying the preconditioner, thus improving its theoretical complexity [40].

While these papers develop algorithms with good theoretical complexity, practical implementations of these methods do not exist. In this paper we take

the approach of developing implementable algorithms and demonstrate their performance on graph Laplacian systems derived from both synthetic and real-world graphs.

A key element of our preconditioner is the use of coarse-grid corrections. Multigrid algorithms were first developed as effective preconditioners for large finite element problems on geometric meshes. Since its introduction in [12], algebraic multigrid methods (AMG) have adapted the original multigrid scheme for use in linear systems with no underlying geometry.

The rest of the paper is organized as follows. Section 4.2 defines the graph Laplacian and the problem we solve. Section 4.3 derives our direct hierarchical method. Section 4.4 proves the complexity requirements for the algorithm. Section 4.5 shows how this method can be used to construct a support graph preconditioner, and Section 4.6 improves this preconditioner using a two-grid approach. Finally, Section 4.7 shows experimental results, and Section 4.8 concludes.

4.2 Problem Statement

Suppose we have a graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges (pairs of vertices $(u, v) \in V \times V$), with each edge having an associated weight w_{uv}. We consider undirected graphs, which means that if $(u, v) \in E$, then $(v, u) \in E$ with $w_{vu} = w_{uv}$. In what follows, we assume that G is a connected graph; that is, any two vertices u and u' can be connected by a path of edges $(u_{s-1}, u_s) \in E$, $s = 1, 2, \ldots, m = m(u, u')$, where $u_0 = u$ and $u_m = u'$. Another notation that we use in what follows is $\mathbf{1} = (1)$ being the constant vector with

unit entries; also the ith unit coordinate vector is denoted by $\boldsymbol{e}_i$.

The *graph Laplacian* is a matrix representation of an undirected graph. It is defined as follows:

$$L_{uv} = \begin{cases} -w_{uv} & (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$
$$L_{uu} = \sum_{v:(u,v)\in E} w_{uv} \tag{4.1}$$

Note that $L = L^T$ and the sum of each row and each column of L is 0, and so $L\mathbf{1} = L^T\mathbf{1} = \mathbf{0}$.

Thus, the problem of interest is to solve the linear system

$$L\boldsymbol{x} = \boldsymbol{b} \tag{4.2}$$

for $\boldsymbol{x}$, where $\mathbf{1}^T\boldsymbol{b} = 0$.

In Section 4.3, we focus on connected graphs that have a p-cut of η edges, with each component $G_1, ..., G_p$ of roughly equal size. Let $V_1, ..., V_p$ denote the vertices of the components, and $n_i = |V_i|$. Let C denote the set of edges of the cut, so that $\eta = |C|$.

4.3 Derivation

4.3.1 A Two-Level Hierarchy

The graph Laplacian is a singular matrix. First, we modify the problem to an equivalent but invertible one.

Lemma 6. *Given a connected graph Laplacian L, some non-zero vector $\boldsymbol{e}$ with non-negative entries, and some $w > 0$, let*

$$\mathcal{L} = L + w\boldsymbol{e}\boldsymbol{e}^T. \tag{4.3}$$

Then

1. *$\mathcal{L}$ is invertible.*
2. *$\mathcal{L}$ is positive-definite.*
3. *If $\mathbf{1}^T\boldsymbol{b} = 0$ and $\mathcal{L}\boldsymbol{x} = \boldsymbol{b}$, then $L\boldsymbol{x} = \boldsymbol{b}$ as well.*

Proof. To prove (1), suppose $\mathcal{L}$ is singular. Then there exists a vector $\boldsymbol{x}$ such that

$$L\boldsymbol{x} = -w\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{x}.$$

The only null vector of the left-hand side of this equation is $\mathbf{1}$, which is not a null vector of the right-hand side, so $L\boldsymbol{x} \neq \mathbf{0}$. So, for the above relation to hold, we must have $L\boldsymbol{x} \propto \boldsymbol{e}$. But $\boldsymbol{e} \notin \text{range}(L)$, as it is not orthogonal to $\mathbf{1}$. Thus, there is no such vector $\boldsymbol{x}$.

To show (2), Note that

$$\boldsymbol{x}^T\mathcal{L}\boldsymbol{x} = \boldsymbol{x}^T L\boldsymbol{x} + w(\boldsymbol{e}^T\boldsymbol{x})^2.$$

Both terms on the right-hand side are at least zero, so $\mathcal{L}$ is positive semidefinite. By (1), $\mathcal{L}$ is invertible, and so is positive definite.

To prove (3), note that

$$0 = \mathbf{1}^T \boldsymbol{b} = \mathbf{1}^T \left(L + w\boldsymbol{e}\boldsymbol{e}^T \right) \boldsymbol{x} = \mathbf{0}^T \boldsymbol{x} + w(\mathbf{1}^T \boldsymbol{e})(\boldsymbol{e}^T \boldsymbol{x}).$$

The factor $\mathbf{1}^T \boldsymbol{e} \neq 0$, so the above relation holds if and only if $\boldsymbol{e}^T \boldsymbol{x} = 0$. Therefore,

$$L\boldsymbol{x} = \mathcal{L}\boldsymbol{x} = \boldsymbol{b}.$$

□

Consider the block-diagonal matrix

$$\hat{L} = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_p \end{bmatrix}. \tag{4.4}$$

Then L has the form

$$L = \hat{L} + \sum_{(u,v) \in C} w_{uv} \boldsymbol{d}_{uv} \boldsymbol{d}_{uv}^T, \tag{4.5}$$

where $\boldsymbol{d}_{uv} = \boldsymbol{e}_u - \boldsymbol{e}_v$. Note that for $\mathcal{L} = L + w\boldsymbol{e}_{u_p} \boldsymbol{e}_{u_p}^T$, we have

$$\mathcal{L} = \hat{L} + w\boldsymbol{e}_{u_p} \boldsymbol{e}_{u_p}^T + \sum_{(u,v) \in C} w_{uv} \boldsymbol{d}_{uv} \boldsymbol{d}_{uv}^T.$$

We choose $w = 1$.

Now, we wish to solve for $\boldsymbol{x}$ in Equation (4.2). We will do this by first solving a set of smaller systems of equations on each of the $L_1, ..., L_p$. But we want each of these sub-systems to be invertible as well. So, for each $j = 1, ..., p$, let u_j

denote the index of a vertex in V_j. Then

$$\begin{aligned}\mathcal{L} &= (\hat{L} + \sum_{j=1}^{p} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T) - \sum_{j=1}^{p-1} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T \\ &= \hat{\mathcal{L}} \quad - \sum_{j=1}^{p-1} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T \\ &= \hat{\mathcal{L}} \quad + F^T W F, \end{aligned} \tag{4.6}$$

where

$$F = [E\ D] \tag{4.7}$$

$$E = \left[\boldsymbol{e}_{u_1}\ \cdots\ \boldsymbol{e}_{u_{p-1}}\right] \tag{4.8}$$

$$D = [\cdots\ \boldsymbol{d}_{uv}\cdots]_{(u,v)\in C} \tag{4.9}$$

$$W = \text{diag}(-T, S) \tag{4.10}$$

$$T = \text{diag}(\mathbf{1}) \tag{4.11}$$

$$S = \text{diag}(\cdots, w_{u,v}, \cdots). \tag{4.12}$$

Note that the matrix $\hat{\mathcal{L}}$ is block-diagonal of the form

$$\hat{\mathcal{L}} = \begin{bmatrix} L_1 + \boldsymbol{e}_{u_1}\boldsymbol{e}_{u_1}^T & & \\ & \ddots & \\ & & L_p + \boldsymbol{e}_{u_p}\boldsymbol{e}_{u_p}^T \end{bmatrix},$$

where each block has the same form as in Equation (4.3). Hence each block of $\hat{\mathcal{L}}$ is itself an invertible matrix, by Lemma 6, so $\hat{\mathcal{L}}$ is invertible.

Lemma 7. *The matrix*

$$-T^{-1} + E^T\left(\hat{\mathcal{L}} + DSD^T\right)^{-1} E \tag{4.13}$$

is symmetric negative-definite.

Proof. As shown above,

$$\mathcal{L} = \hat{\mathcal{L}} + D^T S D - E^T T E$$

is symmetric positive-definite. Then so is

$$\left(\hat{\mathcal{L}} + D^T S D\right)^{-1/2} \mathcal{L} \left(\hat{\mathcal{L}} + D^T S D\right)^{-1/2} = I - RR^T$$

where

$$R = \left(\hat{\mathcal{L}} + D^T S D\right)^{-1/2} E^T T^{1/2}.$$

Then

$$I - R^T R = I - T^{1/2} E^T \left(\hat{\mathcal{L}} + D^T S D\right)^{-1} E T^{1/2}$$

is also symmetric positive definite. Pre- and post-multiplication by $T^{-1/2}$ proves the result. □

Theorem 5.

$$\mathcal{L}^{-1} = \hat{\mathcal{L}}^{-1} + \hat{\mathcal{L}}^{-1} F \left(W^{-1} + F^T \hat{\mathcal{L}}^{-1} F\right)^{-1} F^T \hat{\mathcal{L}}^{-1}. \tag{4.14}$$

Proof. We must show that $W^{-1} + F^T \hat{\mathcal{L}}^{-} F$ is invertible. Note that

$$W^{-1} + F^T \hat{\mathcal{L}}^{-1} F = \begin{bmatrix} -T^{-1} + E^T \hat{\mathcal{L}}^{-1} E & E^T \hat{\mathcal{L}}^{-1} D \\ D^T \hat{\mathcal{L}}^{-1} E & S^{-1} + D^T \hat{\mathcal{L}}^{-1} D \end{bmatrix}. \tag{4.15}$$

The lower-right block is symmetric positive-definite, hence invertible. The respective Schur complement is

$$-T^{-1} + E^T \left(\hat{\mathcal{L}}^{-1} - \hat{\mathcal{L}}^{-1} D \left(S^{-1} + D^T \hat{\mathcal{L}}^{-1} D\right)^{-1} D^T \hat{\mathcal{L}}^{-1}\right) E.$$

Note that $\hat{\mathcal{L}}, S$, and $S^{-1} + D^T \hat{\mathcal{L}}^{-1} D$ are all invertible. We may therefore apply the Sherman-Morrison-Woodbury formula (see, e.g. Proposition 3.5 of [73]) to rewrite the above as

$$-T^{-1} + E^T \left(\hat{\mathcal{L}} + DS D^T\right)^{-1} E.$$

Lemma 7 shows us that this matrix is negative definite and therefore invertible.

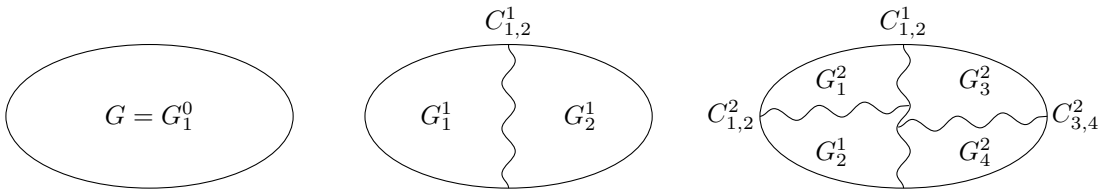


Figure 4.1: Three different views of the hierarchically decomposed graph G. Here $G = (V, E)$ is decomposed as $G = (V_1^1 \cup V_2^2, E_1^1 \cup E_2^1 \cup C_{1,2}^1)$, and $G_1^1 = (V_1^1, E_1^1)$ and $G_2^1 = (V_2^1, E_2^1)$ are decomposed similarly.

Thus, both the lower-right block of Equation (4.15) and its Schur complement are invertible, proving that $W^{-1} + F^T \hat{\mathcal{L}}^{-1} F$ is invertible. Therefore, we may again apply the Sherman-Morrison-Woodbury formula to Equation (4.6) to obtain the result. □

4.3.2 Multilevel Hierarchies

We can modify this decomposition to allow it to handle much larger cuts by imposing some extra restrictions on the structure of the cut. In particular, suppose the graph can be recursively cut into p components such that each p-cut consists of at most η edges. Then we may apply our decomposition recursively, decomposing each sub-problem using another call to the decompose function.

This creates a hierarchy of subgraphs. Let G_i^ν denote subgraph i at hierarchy level ν, for $\nu = 0, ..., \nu_M$, where $\nu_M \simeq O(\log n)$. So $G = G_1^0$, and in Equaton 4.4, each L_i refers to G_i^1. We may globally index all level-ν subgraphs as $G_1^\nu, ..., G_{p^\nu}^\nu$, or we may refer to the children of G_i^ν as $G_{i,1}^\nu, ..., G_{i,p}^\nu$. See Figure 4.1 for a diagram of this hierarchy.

Define graph Laplacians L_i^ν, vertex sets V_i^ν, subgraph edge sets E_i^ν similarly.

We define $C^{\nu}_{i,j}$ to be the level-ν edge cut between G^{ν}_i and G^{ν}_j, or C^{ν} to be the total level-ν edge cut.

Algorithms for graph decomposition as well as solution are shown in Algorithms 9 and 10.

Algorithm 9: Construction of HDecomp Object

---

function HATLINV($\boldsymbol{b}; H.P, \nu$)
  Split $\boldsymbol{b}$ into subvectors $\boldsymbol{b}_1, ..., \boldsymbol{b}_p$.
  for $j = 1, ..., p$ **do**
    $\boldsymbol{x}_j \leftarrow$ HSOLVE($H.P[j], \boldsymbol{b}_j$)
  end for
  return $\left[x_1, ..., x_p\right]$.
end function
procedure HDECOMP(G^{ν}_i, ν)
  Initialize object H
  Find $G^{\nu+1}_{i,1}, ..., G^{\nu+1}_{i,p}$.
  for subgraph $G^{\nu+1}_{i,j}$, $j = 1, ..., p$ **do**
    $H.P[j] \leftarrow$ HDECOMP($G^{\nu+1}_{i,j}, \nu + 1$)
  end for
  $\widehat{L}^{-1} \leftarrow$ HATLINV($\cdot\,; H.P, \nu + 1$).
  Construct F as in (4.7).
  Compute $\widehat{F} \leftarrow \widehat{\mathcal{L}}^{-1}F$.
  Compute and factor $\widehat{W} \leftarrow W^{-1} + F^T\widehat{F}$.
  Return $\widehat{L}^{-1}, \widehat{W}, F, \widehat{F}$.
end procedure

Algorithm 10: Hierarchical Solve

Require: $\mathbf{1}^T \boldsymbol{b} = 0$.

procedure HSOLVE($H, \boldsymbol{b}$)
  if $\boldsymbol{b} = 0$ **then**
    return $\mathbf{0}$.
  end if
  $\widehat{\mathcal{L}}^{-1}, \widehat{W}^{-1}, F, \widehat{F} \leftarrow H$.
  Recursively compute $\tilde{\boldsymbol{x}} \leftarrow \widehat{\mathcal{L}}^{-1}\boldsymbol{b}$.
  Solve $\boldsymbol{y} \leftarrow \widehat{W}^{-1}F^T\tilde{\boldsymbol{x}}$.
  $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}} - \widehat{F}\boldsymbol{y}$.
  return $\boldsymbol{x}$.
  if top of hierarchy **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x} - n^{-1}\mathbf{1}\mathbf{1}^T\boldsymbol{x}$.
  end if
end procedure

4.4 Complexity

In this section we derive the time and space complexity of the hierarchical decomposition algorithm. Assume that at each level of the hierarchy, we partition the graph into at most p partitions with an edge cut of size at most η.

4.4.1 Solve Time Complexity

For Dense $\boldsymbol{b}$

We consider here the case in which the decomposition has already been computed at for each of the $\log_p n$ levels of the hierarchy, and we wish to find an $\boldsymbol{x}$ such that $\mathcal{L}\boldsymbol{x} = \boldsymbol{b}$ for some $\boldsymbol{b}$ such that $\mathbf{1}^T\boldsymbol{b} = 0$. In the following, we use $\widehat{W}$ as defined in Algorithm 9.

Suppose we have already computed $\tilde{\boldsymbol{x}} = (\hat{\mathcal{L}}_i^\nu)^{-1}\boldsymbol{b}$, and we wish to compute $(\mathcal{L}_i^\nu)^{-1}\boldsymbol{b}$. The amount of time required for this computation is given by the following steps:

1. Compute $\boldsymbol{b}' = F^T\tilde{\boldsymbol{x}}$. The matrix F has special sparse structure (see (4.7)), so this takes time $\eta + p - 1$.
2. Solve a linear system $\widehat{W}\boldsymbol{y} = \boldsymbol{b}'$. Assuming we have already factored $\widehat{W}$, this takes time at most $(\eta + p - 1)^2$.
3. Compute $\boldsymbol{y}' = \hat{F}\boldsymbol{y}$. The matrix $\hat{F}$ is in general dense, and has size $n_\nu \times \eta + p - 1$. So, this takes time $n_\nu(\eta + p - 1)$.
4. Add the result as $\boldsymbol{x} = \tilde{\boldsymbol{x}} + \boldsymbol{y}'$. This takes time n_ν.

So, this step requires time $O(n_\nu(\eta + p))$. But at the ν'th level of the hierarchy, $n_\nu = n/p^\nu$, so in total this step takes time $O(np^{-\nu}(\eta + p))$.

Now we wish to do this for each $i = 1, ..., p^\nu$ in this level of the hierarchy. Thus, the time required to solve $\mathcal{L}_i^\nu\boldsymbol{x} = \boldsymbol{b}$ for all i is

$$O\left(p^\nu \frac{n}{p^\nu}(\eta + p)\right) = O(n(\eta + p)). \tag{4.16}$$

Note that this time is independent of the hierarchy level ν. We must do this for each level of the hierarchy, of which there are $\log_p n$, for a final solve time complexity of

$$O((\eta + p)n \log_p n). \tag{4.17}$$

For Sparse $\boldsymbol{b}$

If $\boldsymbol{b}$ has only a small number of nonzero entries, the problem is easier. Suppose that $\boldsymbol{b}$ has only κ nonzero entries. Then at each hierarchy level ν, $\boldsymbol{b}_i^\nu = \mathbf{0}$ for all but at most κ of the indices i. The solution on these sub-vectors is simply $\mathbf{0}$.

So, at the ν'th level of the hierachy, instead of needing to compute p^ν inverses $(\mathcal{L}_i^\nu)^{-1}\boldsymbol{b}_i^\nu$, we need only compute κ of them. Thus, the time requirement at each level of the hierarchy is not $O((\eta + p)n)$, but

$$O\left((\eta + p)\kappa\frac{n}{p^\nu}\right). \tag{4.18}$$

We must do this for each of the $\log_p n$ levels of the hierarchy. However, at each hierarchy level ν, there are only at most κ node sets V_i^ν for which $\boldsymbol{b}$ has nonzero values. Therefore, the time requirement at each hierarcy level ν is given by (4.18), and so the total time complexity is

$$\begin{aligned} O\left(\sum_{\nu=1}^{\log_p n} \kappa(\eta + p)np^{-\nu}\right) &\leq O\left((\eta + p)n\kappa \sum_{\nu=1}^{\infty} p^{-\nu}\right) \\ &= O\left((\eta + p)n\kappa\right) \end{aligned} \tag{4.19}$$

4.4.2 Decomposition Time Complexity

Now we derive the time complexity associated with computing the decomposition at each level. We do not propose a particular algorithm or time complexity associated with the partitioning step; we simply refer to the time required to partition a graph as $\psi(n, p)$. A number of efficient partitioning algorithms and software packages are available, such as METIS [35] or SCOTCH [16]. Note that we also assume that whatever partitioning algorithm is used it successfully finds an edge cut of size at most η.

Construction of the hierarchy occurs in a top-down fashion, followed by computation of $\widehat{F}_i^\nu$ and computation and factoring of $\widehat{W}$, which occurs in a bottom-up fashion. For the top-down phase, we simply call whatever partitioning algorithm we use. At level ν in the hierarchy, each call costs $\psi(n/p^\nu, p)$, and there are p^ν of them to perform. Thus, the cost of the top-down phase is

$$O\left(\sum_{\nu=1}^{\log n} p^\nu \psi(n/p^\nu, p)\right) \tag{4.20}$$

If the partitioning method takes time $O(n)$, then this step takes $O(n \log_p n)$ time.

Now, for the bottom-up stage. Assume that the decomposition has been completed for all hierarchy levels beyond ν. We wish to compute $\widehat{F}_i^\nu = \widehat{\mathcal{L}}^{-1} F$. Because each column of F_i^ν is sparse with $\kappa \leq 2$, computing each column of $\widehat{F}_i^\nu$ requires the sparse solve time $O((\eta + p)np^{-\nu})$ (see (4.17)). There are $(\eta + p)$ such vectors to compute, so the computation costs time $O((\eta + p)^2 np^{-\nu})$.

We must then compute and factor $\widehat{W}$. The matrix W is diagonal of size $\eta + p - 1$, so the computation of W^{-1} requires time $O(\eta + p - 1)$. The computation $(F_i^\nu)^T \widehat{F}_i^\nu$ requires constant time for each entry, because F_i^ν is sparse, for a total of

$O((\eta + p)^2)$. Therefore, the construction of $\widehat{W}$ requires time

$$O\left((\eta + p)^2 np^{-\nu} + (\eta + p)^2\right) = O\left((\eta + p)^2 np^{-\nu}\right).$$

The factorization of $\widehat{W}$ then takes time $O((\eta+p)^3)$. Thus the total computation time is

$$O\left((\eta + p)^2 np^{-\nu} + (\eta + p)^3\right). \tag{4.21}$$

There are p^ν such decompositions to compute at for level ν, resulting in a time of

$$O\left((\eta + p)^2 n + (\eta + p)^3 p^\nu\right) \tag{4.22}$$

for level ν.

We must do this for each of the $\log_p n$ levels, resulting in a total time complexity of

$$\begin{aligned} O\left(\sum_{\nu=1}^{\log_p n} (\eta + p)^2 n + (\eta + p)^3 p^\nu\right) =& O\left((\eta + p)^2 n \log n + (\eta + p)^3 \sum_{\nu=1}^{\log_p n} p^\nu\right) \\ =& O\left((\eta + p)^2 n \log n + (\eta + p)^3 \frac{p^{1+\log_p n} - 1}{p - 1}\right) \\ =& O\left((\eta + p)^2 n \log n + (\eta + p)^3 n\right). \end{aligned} \tag{4.23}$$

If we assume constant η and p, then this simplifies to

$$O(n \log n). \tag{4.24}$$

4.4.3 Storage Complexity

At level ν of the hierarchy, we must store $\hat{F}$ and the factorization of $\widehat{W}$ for each partition. The matrix $\hat{F}$ takes storage of size $(\eta + p)n/p^\nu$, and the factorization of

$\widehat{W}$ takes storage of size $(\eta + p)^2$. Level ν of the hierarchy has p^ν such partitions, and so each level requires

$$O\left(p^\nu((\eta + p)np^{-\nu} + (\eta + p)^2)\right) = O(n(\eta + p) + p^\nu(\eta + p)^2)$$

storage. As there are $\log n$ levels, the total storage complexity is

$$O\left(\sum_{\nu=1}^{\log_p n} n(\eta + p) + p^\nu(\eta + p)^2\right) = O\left((\eta + p)n \log n + (\eta + p)^2 n\right).$$

4.5 Preconditioning

When a graph of interest does not have a hierarchy of p-cuts of size η for acceptably small p and η, we can can still use the above method to construct a preconditioner for the associated linear system. Of the original graph $G = (V, E)$, we build a *support graph* $G_S = (V, E_S)$ where $E_S \subset E$ such that G_S has the desired hierarchical structure. Also let $G_O = (V, E_O)$, where $E_O = E \setminus E_S$.

Now, let L be the graph Laplacian matrix of G, L_S the graph Laplacian of G_S, and L_O the graph Laplacian of G_O. Consider the use of L_S as a preconditioner for L. We would typically write the associated stationary iteration as

$$\boldsymbol{x}_t = (I - \omega^{-1}L_S^{-1}L)\boldsymbol{x}_{t-1} + \omega^{-1}L_S^{-1}\boldsymbol{b} \tag{4.25}$$

for some parameter ω. However, as L_S is a singular matrix, we need to define what we mean by L_S^{-1}. Note that L and L_S have the same null space, $\{\alpha\mathbf{1}, \alpha \in \mathbb{R}\}$, and so L_S is invertible on range(L). The solution to any linear system $L_S\boldsymbol{x} = \boldsymbol{b}$ is only determined up to a constant, so to define it uniquely we select the solution such that $\boldsymbol{x}^T\mathbf{1} = 0$.

With the above definition of L_S^{-1}, i.e., $L_S^{-1} = L_S^{\dagger}$, we wish to ensure that (4.25) is a convergent iteration. To that end, note that the eigenvalues and eigenvectors of $L_S^{\dagger} L$ are the same as that of the generalized eigenvalue problem

$$\lambda L_S \boldsymbol{x} = L\boldsymbol{x}. \tag{4.26}$$

Therefore, the eigenvalues of the error iteration matrix $\mathcal{E} = I - \omega^{-1} L_S^{\dagger} L$ are

$$\tilde{\lambda}_i = 1 - \omega^{-1}\lambda_i, \tag{4.27}$$

where λ_i is an eigenvalue of Equation (4.26). Thus, the iteration is convergent as long as $|\tilde{\lambda}_i| < 1$ for all i.

Now, for an eigenvalue λ_i, we have

$$\lambda_i = \frac{\langle \boldsymbol{x}^i, L\boldsymbol{x}^i \rangle}{\langle \boldsymbol{x}^i, L_S \boldsymbol{x}^i \rangle} = 1 + \frac{\langle \boldsymbol{x}^i, L_O \boldsymbol{x}^i \rangle}{\langle \boldsymbol{x}^i, L_S \boldsymbol{x}^i \rangle}. \tag{4.28}$$

Graph Laplacians are positive semidefinite, and so we see that $\lambda_i \geq 1$ for all i.

Therefore, $|\tilde{\lambda}_i| < 1 \ \forall i$ if and only if $\omega > \lambda_1/2$, where λ_1 is the largest eigenvalue of Equation (4.26).

For small-to-medium sized graphs, this preconditioner is often highly effective on its own. However, for larger graphs, this may not be the case.

To illustrate why, consider Figure 4.2, which shows the top part of the spectrum for the preconditioned matrix $L_S^{\dagger} L$ in solid blue lines for a few example networks. The graphs used are the Epinions1 and Slashdot0811 networks from the Stanford Large Network Database [41], as well as a 256×256 2D grid. As can be seen, the eigenvalues tend to follow a distrbution that is visually similar to a power law, where the vast majority of the eigenvalues are quite close to 1, but a significant number of large eigenvalues still exist.

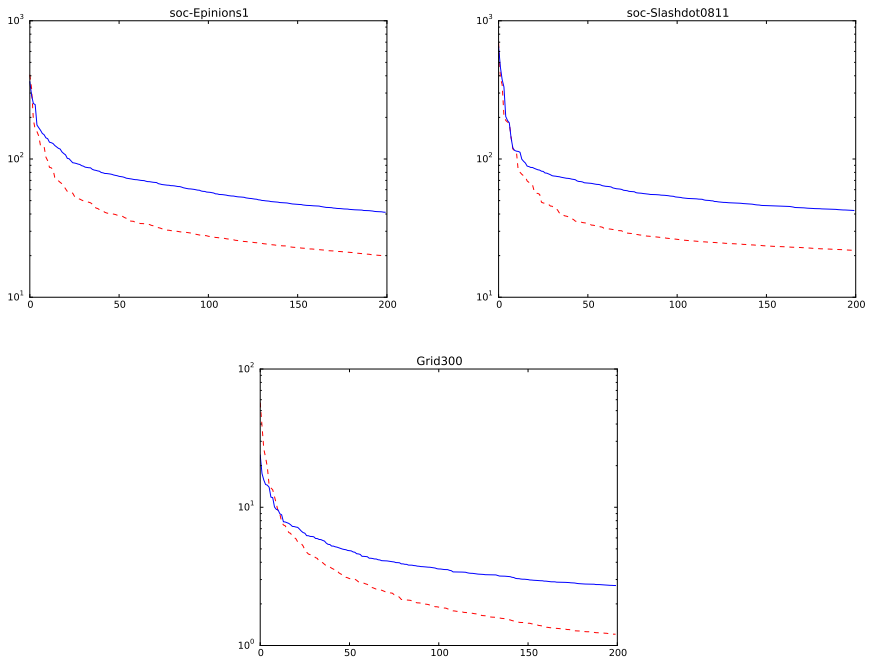


Figure 4.2: The blue line shows the first 200 eigenvalues for the matrix $L_S^{\dagger} L$ on the Epinions1 and Slashdot0811 networks with a maximum atomic subset size of 256, as well as on a 256×256 2D grid with a maximum atomic subset size of 64. The dotted red lines show the top eigenvalues of restricting to vectors that are constant on atomic subsets; that is, the square roots of eigenvalues of $H^T H$ in Equation (4.31).

As can be seen here, the largest eigenvalue can be quite large. In fact, suppose that the initial cut bisecting G has n^{β} edges in it, for some $0 < \beta < 1$. There are only η such edges in the support graph G_S, so if

$$\boldsymbol{x} = \left[\mathbf{1}^T \ -\mathbf{1}^T\right]^T,$$

with all 1's on one side of the cut and all -1's on the other, then

$$\frac{\langle \boldsymbol{x}, L\boldsymbol{x} \rangle}{\langle \boldsymbol{x}, L_S \boldsymbol{x} \rangle} \propto n^{\beta}.$$

In 3D grids, $\beta = 2/3$, and in social networks, we expect β to typically be near 1. This suggests that L_S by itself is not an effective preconditioner for social

networks. In the following section, we show how these damaging error modes can often be mitigated by using a two-grid preconditioning scheme.

4.6 Two-Grid Preconditioning

As discussed above, a few large eigenvalues of $L_S^\dagger L$ tend to damage the condition number of the system. We can avoid much of this problem through the use of two-grid preconditioning.

4.6.1 A Two-Grid Preconditioner

Elements of a Multilevel Preconditioner

Traditional algebraic multigrid makes use of two ingredients: a *smoother* and a *coarse-grid correction*. The smoother is a preconditioner M^{-1} whose application tends to most effectively shrink those error modes associated with the largest eigenvalues of $M^{-1}L$, such as a Jacobi or Gauss-Seidel iteration.

The coarse-grid correction, on the other hand, shrinks those error modes associated with the smallest eigenvalues (the *algebraically smooth* modes) of $M^{-1}L$. This is done by defining an interpolation operator $P \in \mathbb{R}^{+n \times n_c}$, $n_c \ll n$, that spans a *coarse subspace* and an associated *coarse matrix* $L_c = P^T LP$. Note that we impose P to have nonnegative entries. One then makes a coarse-grid correction by restricting the current residual to the coarse subspace and solving the resulting problem with L_c. Note that if the rows of P sum to 1, then L_c is also a graph Laplacian.

Definition 6 (Solving problems with a coarse graph Laplacian)**.** *Let P be a piecewise constant interpolant such that $P\mathbf{1}_c = \mathbf{1}$ and form $L_c = P^T LP$. For any given $\boldsymbol{b}_c : (\mathbf{1}_c)^T \boldsymbol{b}_c = 0$, we define the unique solution to the coarse problem*

$$L_c \boldsymbol{x}_c = \boldsymbol{b}_c,$$

as follows. We select $\boldsymbol{x}_c$ such that $(\mathbf{1})^T P\boldsymbol{x}_c = 0$, which is equivalent to $(\mathbf{1}_c)^T (P^T P)\boldsymbol{x}_c = 0$, or $(P\boldsymbol{x}_c)^T (P\mathbf{1}_c) = 0$. We use the notation $\boldsymbol{x}_c = L_c^\dagger \boldsymbol{b}_c$.

The traditional combined three-step AMG preconditioner reads as follows:

1. Apply smoother:
$$\boldsymbol{x}_{t+\frac{1}{3}} = (I - M^{-1}L)\boldsymbol{x}_t + M^{-1}\boldsymbol{b}.$$
2. Apply coarse-grid correction:
$$\boldsymbol{x}_{t+\frac{2}{3}} = (I - PL_c^\dagger P^T L)\boldsymbol{x}_{t+\frac{1}{3}} + PL_c^\dagger P^T \boldsymbol{b}.$$
3. Apply smoother (in a symmetric fashion):
$$\boldsymbol{x}_{t+1} = (I - M^{-T}L)\boldsymbol{x}_{t+\frac{2}{3}} + M^{-T}\boldsymbol{b}.$$

This results in an error iteration matrix

$$\mathcal{E} = (I - M^{-1}L)(I - PL_c^\dagger P^T L)(I - M^{-1}L). \tag{4.29}$$

The (weighted) Jacobi smoother M performs local updates only. This leaves global error components which can be targeted by a coarse-grid correction. In our case, L_S is a global operator (similar to L) rather than a local operator. After applying $L_S^\dagger$, most of the error is eliminated, but, as we argue in Section 4.6.1, still there are error components that can be well approximated by a coarse-grid correction. We note that in these two cases we are looking at the spectra of two different operators, $L_S^\dagger L$ and $M^{-1}L$.

A Coarse Subspace

Assume that the nodes in G are ordered according to their hierachical decomposition as described in Section 4.3. Let

$$\boldsymbol{\ell}_i = \begin{bmatrix} \mathbf{0}^T & \cdots & \mathbf{0}^T & \mathbf{1}^T & \mathbf{0}^T & \cdots & \mathbf{0}^T \end{bmatrix}^T,$$

that is, the vector with ones on the nodes $V_i^{\nu_M}$ of an atomic subset, and zeroes everywhere else. Recalling that ν_M is the deepest level of the bisection hierarchy of Section 4.3, consider the matrix

$$\tilde{P} = \begin{bmatrix} \sqrt{n_{\nu_M,1}^{-1}}\boldsymbol{\ell}_1 & \cdots & \sqrt{n_{\nu_M,p^{\nu_M}}^{-1}}\boldsymbol{\ell}_{p^{\nu_M}} \end{bmatrix}. \tag{4.30}$$

Let

$$H = L_S^{\dagger} L\tilde{P} \tag{4.31}$$

In Figure 4.2, we show the square roots of the first 200 eigenvalues of the matrix $H^T H$ for each network in dashed red lines. This figure shows that these $\tilde{P}$ approximately span the invariant subspace associated with the largest eigenvalues of $L_S^{\dagger} L$.

So, define

$$P = \begin{bmatrix} \boldsymbol{\ell}_1 & \cdots & \boldsymbol{\ell}_{p^{\nu_M}} \end{bmatrix}. \tag{4.32}$$

This matrix is the same as $\tilde{P}$ except all its nonzero entries are 1. Then span(P) is our coarse subspace of dimension $n_c = p^{\nu_M}$. Note that, if one wishes to use a smaller coarse grid, one can simply take the subsets associated with another hierarchy level ν.

A Two-Grid Preconditioner

Unlike Jacobi or Gauss-Seidel smoothing, because $\sigma(L_S^\dagger L) \geq 1$, L_S is not convergent as a stationary iteration without weighting the iteration in such a way that would limit its effectiveness on the lowest-eigenvalue modes. This problem gets even worse if $L_S^\dagger$ is used as M^{-1} in the traditional AMG method of Section 4.6.1, as Equation (4.29) shows that the eigenvalues of $(I - L_S^\dagger L)$ would be squared. So, rather than placing the coarse-grid correction in the middle of the two-grid preconditioner as in classical algebraic multigrid, we place L_S in the middle. This leads to the three-step preconditioner

1. Apply coarse-grid correction:

$$\boldsymbol{x}_{t+\frac{1}{3}} = (I - PL_c^\dagger P^T L)\boldsymbol{x}_t + PL_c^\dagger P^T \boldsymbol{b}.$$

2. Apply L_S:

$$\boldsymbol{x}_{t+\frac{2}{3}} = (I - L_S^\dagger L)\boldsymbol{x}_{t+\frac{1}{3}} + L_S^\dagger \boldsymbol{b}.$$

3. Apply coarse-grid correction:

$$\boldsymbol{x}_{t+1} = (I - PL_c^\dagger P^T L)\boldsymbol{x}_{t+\frac{2}{3}} + PL_c^\dagger P^T \boldsymbol{b}.$$

First of all, we note that the above algorithm is well-defined, namely, the actions of $L_c^\dagger$, $L_c^\dagger P^T L$, and $L_S^\dagger$, $L_S^\dagger L$, are well-defined, since $\mathbf{1}_c^T(P^T L) = (P\mathbf{1}_c)^T L = \mathbf{1}^T L = 0$ (see Definition 6).

The above algorithm has the following property: if $\mathbf{1}^T \boldsymbol{x}_t = 0$, then also $\mathbf{1}^T \boldsymbol{x}_{t+s} = 0$ for $s = 1/3, 2/3, 1$. This is due to the fact that $\mathbf{1}^T P L_c^\dagger = \mathbf{1}_c^T (P^T P) L_c^\dagger = 0$ and $\mathbf{1}^T L_S^\dagger = 0$.

The above algorithm results in an error iteration matrix

$$\mathcal{E} = (I - PL_c^\dagger P^T L)(I - L_S^\dagger L)(I - PL_c^\dagger P^T L). \tag{4.33}$$

We do not claim that E has a norm less than one (in fact, we have $E\mathbf{1} = \mathbf{1}$), rather we will use this expression (or the algorithm above) to define a preconditioner B^{-1}.

4.6.2 Analysis of the Two-Grid Preconditioner

We wish to develop the tools to analyze the rate of convergence of this preconditioned system. Traditionally, a preconditioner is described by a matrix B, but the action of the preconditioner is through the application of B^{-1}. The convergence rate of a preconditioned conjugate gradient iteration is then determined by $\sqrt{\kappa_2/\kappa_1}$ ([7]), where

$$\kappa_1 B \preceq L \preceq \kappa_2 B.$$

In our case, B is not known a priori, and so we study it by deriving B^{-1} from the above algorithm. We can do this by assuming $E = I - B^{-1}L$ and writing (4.33) as

$$\begin{aligned}
\mathcal{E} =& (I - PL_c^\dagger P^T L)(I - L_S^\dagger L)(I - PL_c^\dagger P^T L) \\
=& (I - PL_c^\dagger P^T L)(I - PL_c^\dagger P^T L) - (I - PL_c^\dagger P^T L)L_S^\dagger L(I - PL_c^\dagger P^T L) \\
=& I - PL_c^\dagger P^T L - (I - PL_c^\dagger P^T L)L_S^\dagger (I - LPL_c^\dagger P^T)L \\
=& I - B^{-1}L,
\end{aligned}$$

where

$$B^{-1} = PL_c^\dagger P^T + (I - PL_c^\dagger P^T L)L_S^\dagger (I - LPL_c^\dagger P^T).$$

We notice now that if $\boldsymbol{b}$ is such that $\mathbf{1}^T\boldsymbol{b} = 0$, then $B^{-1}\boldsymbol{b}$ is well-defined. Indeed, since $\mathbf{1}_c{}^T P^T\boldsymbol{b} = \mathbf{1}^T\boldsymbol{b} = 0$, $L_c^\dagger P^T\boldsymbol{b}$ is well-defined. Also, $L_S^\dagger\boldsymbol{b}$ and $L_S^\dagger L$ are well-defined. Similarly, $\mathbf{1}_c{}^T P^T L = \mathbf{1}^T L = 0$, hence $L_c^\dagger P^T L$ is well-defined as well. An additional property of B^{-1} is that $\mathbf{1}^T B^{-1}\boldsymbol{b} = 0$.

We have that $L^\dagger$ is symmetric positive semi-definite, hence $(L^\dagger)^{\frac{1}{2}}$ is well-defined as a symmetric positive semi-definite matrix. The following idenity holds:

$$L(L^\dagger)^{\frac{1}{2}} = L^{\frac{1}{2}}.$$

Now, consider

$$L^{\frac{1}{2}}\mathcal{E}(L^\dagger)^{\frac{1}{2}} = (I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})(I - L^{\frac{1}{2}}L_S^\dagger L^{\frac{1}{2}})(I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}}.$$

We also have,

$$L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}} - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}} = L^{\frac{1}{2}}\mathcal{E}(L^\dagger)^{\frac{1}{2}}.$$

Therefore, for any $\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0$, we have $L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}}\boldsymbol{x} = \boldsymbol{x}$, hence

$$\boldsymbol{x}^T(I - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}})\boldsymbol{x} = \boldsymbol{y}^T(I - L^{\frac{1}{2}}L_S^\dagger L^{\frac{1}{2}})\boldsymbol{y} \leq 0, \tag{4.34}$$

where $\boldsymbol{y} = (I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})\boldsymbol{x}$. That is, the operator $I - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}}$ is negative semi-definite in the subspace $\{\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0\}$. In other words, the operator $L^\dagger - B^{-1}$ is negative semi-definite in the same subspace, which implies that B^{-1} is positive definite (since $L^\dagger$ is positive definite) in the same subspace. Hence, we can define its inverse, B, well-defined in the subspace $\{\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0\}$. More over, we have the inequality

$$B \preceq L. \tag{4.35}$$

We now present a useful lemma.

Lemma 8. *Given two vector spaces V and W contained in some $\mathbb{R}^n$, consider two matrices T and N acting on vectors in $\mathbb{R}^n$. We assume that T is symmetric and it maps V onto itself, whereas N maps V onto W, i.e., for each $\boldsymbol{w} \in W$ there is a $\boldsymbol{v} \in V$ such that $N\boldsymbol{v} = \boldsymbol{w}$. Moreover, we assume that $T - N^T N$ is s.p.d. on V. Consider*

$$Z = N\left(T - N^T N\right)^{-1} N^T.$$

We have that Z is s.p.d. on W, hence invertible on W, and for each $\boldsymbol{w} \in W$,

$$\frac{\boldsymbol{w}^T Z^{-1} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}} = -1 + \min_{\boldsymbol{v}:\, N\boldsymbol{v}=\boldsymbol{w}} \frac{\boldsymbol{v}^T T \boldsymbol{v}}{\boldsymbol{w}^T \boldsymbol{w}}.$$

Proof. This lemma is similar to Lemma 3.1 of [26], as is its proof. For completeness we present its proof here.

We first show that if $N^T \boldsymbol{w} = 0$ for $\boldsymbol{w} \in W$, then $\boldsymbol{w} = 0$. Indeed, since $\boldsymbol{w} = N\boldsymbol{v}$ for some $\boldsymbol{v} \in V$, we have that $N^T N\boldsymbol{v} = 0$. The latter in particular implies $\boldsymbol{v}^T N^T N\boldsymbol{v} = 0$, i.e., $\|N\boldsymbol{v}\| = 0$ and hence $0 = N\boldsymbol{v} = \boldsymbol{w}$. This implies that Z is invertible, hence s.p.d. on W.

Consider now the constrained minimization problem:

Given $\boldsymbol{w} \in W$ compute

$$\frac{1}{2}\, \boldsymbol{v}^T T \boldsymbol{v} \mapsto \min,$$

subject to $N\boldsymbol{v} = \boldsymbol{w}$.

Forming the Lagrangian $\frac{1}{2}\ \boldsymbol{v}^T T \boldsymbol{v} - \boldsymbol{\lambda}^T(\boldsymbol{w} - N\boldsymbol{v})$, the necessary consitions for minimum give

$$\begin{bmatrix} T & N^T \\ N & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{w} \end{bmatrix}.$$

An equivalent system is

$$\begin{bmatrix} T - N^T N & N^T \\ N & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -N^T \boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}.$$

This system has unique solution since $T - N^T N : \ V \mapsto V$ is s.p.d. and its negative Schur complement $Z = N(T - N^T N)^{-1} N^T : \ W \mapsto W$ is also s.p.d. The solution equals,

$$\begin{aligned} \boldsymbol{v} &= -(T - N^T N)^{-1} N^T (\boldsymbol{\lambda} + \boldsymbol{w}), \\ -Z(\boldsymbol{\lambda} + \boldsymbol{w}) &= \boldsymbol{w}. \end{aligned}$$

That is,

$$\boldsymbol{v} = (T - N^T N)^{-1} N^T Z^{-1} \boldsymbol{w}.$$

Then,

$$\boldsymbol{w}^T Z^{-1} \boldsymbol{w} = -\boldsymbol{w}^T (\boldsymbol{\lambda} + \boldsymbol{w}) = -\boldsymbol{w}^T \boldsymbol{w} - \boldsymbol{w}^T \boldsymbol{\lambda}.$$

On the other hand

$$\boldsymbol{\lambda}^T \boldsymbol{w} = \boldsymbol{\lambda}^T N \boldsymbol{v} = \boldsymbol{v}^T N^T \boldsymbol{\lambda} = \boldsymbol{v}^T (-N^T \boldsymbol{w} - (T - N^T N)\boldsymbol{v}) = -\|\boldsymbol{w}\|^2 - \boldsymbol{v}^T (T - N^T N)\boldsymbol{v}.$$

That is, since $\boldsymbol{v} : \ N\boldsymbol{v} = \boldsymbol{w}$, we have

$$\boldsymbol{w}^T Z^{-1} \boldsymbol{w} = \boldsymbol{v}^T (T - N^T N)\boldsymbol{v} = -\boldsymbol{w}^T \boldsymbol{w} + \boldsymbol{v}^T T \boldsymbol{v}$$

Thus, using the fact that $\boldsymbol{v}$ is the minimizer, we have

$$\frac{\boldsymbol{w}^T Z^{-1} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}} = -1 + \min_{\boldsymbol{v}: \, N\boldsymbol{v} = \boldsymbol{w}} \frac{\boldsymbol{v}^T T \boldsymbol{v}}{\boldsymbol{w}^T \boldsymbol{w}},$$

which complets the proof. □

We now present a formula for the condition number of B with respect to L viewed as s.p.d. operators acting on the subspace $S \equiv \{\boldsymbol{x} : \mathbf{1}^T \boldsymbol{x} = 0\}$. This proof draws from that of Theorem 4.1 in [26].

Theorem 7. *Let $\pi_L = PL_c^\dagger P^T L$, and let S denote the subspace $\{\boldsymbol{x} : \mathbf{1}^T \boldsymbol{x} = 0\}$. Then*

$$B \preceq L \preceq \kappa B. \tag{4.36}$$

over the subspace S, where

$$\kappa = \sup_{\boldsymbol{w}} \frac{((I - \pi_L)\boldsymbol{w})^T L ((I - \pi_L)\boldsymbol{w})}{\boldsymbol{w}^T L_S \boldsymbol{w}}. \tag{4.37}$$

Proof. We first note that π_L is a projection (as an operator acting on S). The same holds for its symmeric version

$$\overline{\pi}_L = L^{\frac{1}{2}} P L_c^\dagger P^T L^{\frac{1}{2}}.$$

The fact that $B \preceq L$ was shown above (see (4.35)). To prove the other direction, use the fact that for s.p.d. operators L and B (as mappings on S) we have

$$\kappa = \sup_{\boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\boldsymbol{v}\|^2}.$$

We note that the above maximum is achieved in the subspace $\{\boldsymbol{v} = (I - \overline{\pi}_L)\boldsymbol{w},\ \boldsymbol{w} \in S\} \subset S$. We have

$$\begin{aligned}
\kappa &= \sup_{\boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\boldsymbol{v}\|^2} && (4.38)\\
&= \sup_{\boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\overline{\pi}_L \boldsymbol{v}\|^2 + \|(I - \overline{\pi}_L)\boldsymbol{v}\|^2} && (4.39)\\
&\leq \sup_{\boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|(I - \overline{\pi}_L)\boldsymbol{v}\|^2} && (4.40)\\
&= \sup_{\boldsymbol{w} = (I - \overline{\pi}_L)\boldsymbol{v},\ \boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\boldsymbol{w}\|^2} && (4.41)\\
&\leq \kappa. && (4.42)
\end{aligned}$$

Using the fact that $(I - \overline{\pi}_L)$ is a projection and Equation (4.34), note that, if $\boldsymbol{v} \in S$,

$$\begin{aligned}
\boldsymbol{v}^T (I - L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}}) \boldsymbol{v} &= ((I - \overline{\pi}_L)\boldsymbol{v})^T \left(I - ((I - \overline{\pi}_L)) L^{\frac{1}{2}} L_S^\dagger L^{\frac{1}{2}} ((I - \overline{\pi}_L)) \right) ((I - \overline{\pi}_L)\boldsymbol{v})\\
&\geq (1 - \kappa) ((I - \overline{\pi}_L)\boldsymbol{v})^T ((I - \overline{\pi}_L)\boldsymbol{v}).
\end{aligned}$$

This implies that for $Z = (I - \overline{\pi}_L)L^{\frac{1}{2}}L_S^{\dagger}L^{\frac{1}{2}}(I - \overline{\pi}_L)$, we have (based on (4.38))

$$\kappa = \sup_{v \in S} \frac{\boldsymbol{v}^T Z \boldsymbol{v}}{\boldsymbol{v}^T (I - \overline{\pi}_L)\boldsymbol{v}}.$$

Now, let $N = (I - \overline{\pi}_L)L^{\frac{1}{2}}$. Then $N^T N = L(I - \pi_L)$.

Let $T = L_S + L(I - \pi_L)$. Then T is symmetric and positive semi-definite. Furthermore, $T - N^T N = L_S$. Therefore,

$$Z = (I - \overline{\pi}_L)L^{\frac{1}{2}}L_S^{\dagger}L^{\frac{1}{2}}(I - \overline{\pi}_L) = N(T - N^T N)^{-1}N^T.$$

To use Lemma (8), introduce the spaces $V = S$ and let $W = \text{range}(I - \overline{\pi}_L) \cap S$. Then Lemma 8 gives

$$\kappa = \sup_{\tilde{\boldsymbol{w}} \in W} \frac{\tilde{\boldsymbol{w}}^T Z \tilde{\boldsymbol{w}}}{\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}} = \sup_{\tilde{\boldsymbol{w}} \in W} \frac{\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}}{\tilde{\boldsymbol{w}}^T Z^{-1} \tilde{\boldsymbol{w}}} = \frac{1}{\inf_{\tilde{\boldsymbol{w}} \in W} \left(-1 + \inf_{\boldsymbol{v}: N\boldsymbol{v} = \tilde{\boldsymbol{w}}} \left(\boldsymbol{v}^T T \boldsymbol{v} / \tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}\right)\right)}.$$

Now, let $\tilde{\boldsymbol{w}} = L^{\frac{1}{2}}\boldsymbol{w}$. Then $(\tilde{\boldsymbol{w}})^T \tilde{\boldsymbol{w}} = \boldsymbol{w}^T L \boldsymbol{w}$, and

$$\begin{aligned} & N\boldsymbol{v} = L^{\frac{1}{2}}\boldsymbol{w} \\ \implies & L^{\frac{1}{2}}\boldsymbol{w} = (I - \overline{\pi}_L)L^{\frac{1}{2}}\boldsymbol{v} \\ \implies & \boldsymbol{w} = (I - \pi_L)\boldsymbol{v}. \end{aligned}$$

Note that $\boldsymbol{w} \in \text{range}(I - \pi_L) \cap S$, which we denote $\overline{W}$. This leads to

$$\kappa^{-1} = \inf_{\boldsymbol{w} \in \overline{W}} \left(\inf_{\boldsymbol{v}: (I - \pi_L)\boldsymbol{v} = \boldsymbol{w}} \frac{\boldsymbol{v}^T T \boldsymbol{v}}{\boldsymbol{w}^T L \boldsymbol{w}} - 1 \right).$$

But

$$\boldsymbol{v}^T T \boldsymbol{v} = \boldsymbol{v}^T L_S \boldsymbol{v} + \boldsymbol{v}^T N^T N \boldsymbol{v} = \boldsymbol{v}^T L_S \boldsymbol{v} + \boldsymbol{w}^T L \boldsymbol{w},$$

so

$$\kappa^{-1} = \inf_{\boldsymbol{w} \in \overline{W}} \inf_{\boldsymbol{v}: (I - \pi_L)\boldsymbol{v} = \boldsymbol{w}} \frac{\boldsymbol{v}^T L_S \boldsymbol{v}}{\boldsymbol{w}^T L \boldsymbol{w}} = \inf_{\boldsymbol{v}: \boldsymbol{v} \in S} \frac{\boldsymbol{v}^T L_S \boldsymbol{v}}{((I - \pi_L)\boldsymbol{v})^T L ((I - \pi_L)\boldsymbol{v})}.$$

But both the numerator and the denominator ignore any components of $\boldsymbol{v}$ in the direction of $S^{\perp} = \{\alpha \mathbf{1}\}$, so we may write this simply as

$$\kappa = \sup_{\boldsymbol{v}} \frac{((I - \pi_L)\boldsymbol{v})^T L ((I - \pi_L)\boldsymbol{v})}{\boldsymbol{v}^T L_S \boldsymbol{v}}. \tag{4.43}$$

This completes the proof. □

Corollary 3. *When used in combination with the preconditioned conjugate gradient method, the number of iterations required when using the two-grid preconditioner of Section 4.6.1 is bounded by $O(\sqrt{\kappa})$, where κ is given in Equation* (4.37).

Remark In this paper we have not provided specific algorithms for constructing the partition hierarchy or for deciding which edges to keep for L_S. We can, however, use the above theorem to develop some guidelines. Decompose L as $L = L_W + L_B$, where L_W is a disconnected graph Laplacian with the edges within the smallest-level atomic subsets $G_i^{\nu_M}$ and L_B is a graph Laplacian with the between-subset edges. Note that all within-atomic subset edges are part of the support graph Laplacian L_S, so L_W is itself a support graph Laplacian of L_S. We can further decompose L_B as $L_B = L_{BS} + L_{BS^{\perp}}$, where L_{BS} contains the edges of L_B that are in L_S, and $L_{BS^{\perp}}$ contains the edges that are not. Note that $L_S = L_W + L_{BS}$.

The coarse grid ignores all within-atomic subset edges. That is, $P^T L_W = \mathbf{0}^T$. Therefore, $L_c = P^T L P = P^T L_B P$. Thus, we may rewrite Equation (4.37) as

$$\kappa = \sup_{\boldsymbol{w}} \frac{((I - \pi_L)\boldsymbol{w})^T L ((I - \pi_L)\boldsymbol{w})}{\boldsymbol{w}^T (L_W + L_{SB})\boldsymbol{w}},$$

where $\pi_L = P(P^T L_B P)^{\dagger} P^T L_B$.

Suppose that

$$\boldsymbol{w}^T L \boldsymbol{w} = \boldsymbol{w}^T (L_W + L_{BS} + L_{BS^{\perp}})\boldsymbol{w} \approx \boldsymbol{w}^T L_{BS^{\perp}} \boldsymbol{w}.$$

In this case, the denominator of Equation (4.37) will be small. Suppose further that $P^T L_B \boldsymbol{w} \approx \mathbf{0}$. This can happen, for example, through "parallel edges" in $L_{BS^\perp}$, that is, multiple edges stretching between the same two atomic subsets. Then the numerator of Equation (4.37) is approximately $\boldsymbol{w}^T L \boldsymbol{w} \approx \boldsymbol{w}^T L_{BS^\perp} \boldsymbol{w}$, which is by assumption not small. That is, we have a small denominator and a large numerator, resulting in a large κ.

So, we would like to construct the partition hierarchy and L_S to avoid this situation. This leads to the following guidelines:

1. The atomic subsets should be as well-connected as possible, so that $\boldsymbol{w}^T L_W \boldsymbol{w}$ is as large as possible.
2. Often, high-degree nodes have more incident edges than can be captured in L_{SB}. Therefore, as much as possible, high-degree nodes and their neighbors should be placed in the same atomic subset.
3. Suppose u and v are two high-degree nodes, are placed in the same atomic subset, and have many neighbors in other atomic subsets. Then the intersection between u's neighboring atomic subsets and v's neighboring atomic subsets should be as small as possible. Otherwise, a $\boldsymbol{w}$ such that $w_u \approx -w_v$ with zeros everywhere else can result in a large κ.

4.7 Experiments

In this section we present experiments to demonstrate the efficiency of the algorithms presented here.

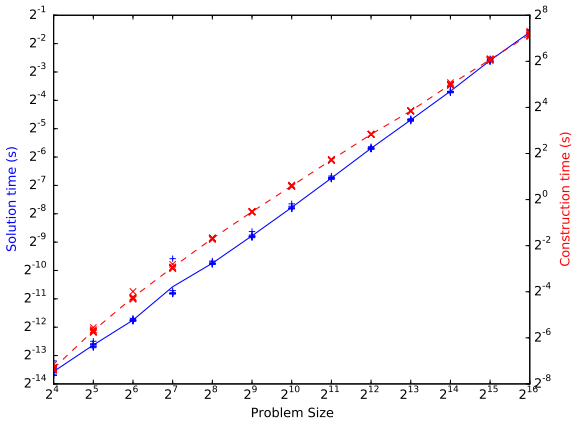


Figure 4.3: Time to construct a solver (red X), and time to perform a solve (blue +) using the direct solver of Section 4.3 on graphs that can be recursively bisected with cuts of size 8. Dotted red and solid blue lines connect sample means of 8 samples each.

4.7.1 Direct Solver Scaling

In this section we demonstrate the scaling behavior of the direct solver described in Section 4.3. We recursively construct a graph Laplacian as follows:

1. If constructing a graph of size $N \leq 8$, add 20 edges uniformly at random. If the resulting graph is not connected, try again.
2. If constructing a graph of size $N > 8$, construct two graphs each of size $N/2$ and connect them with 8 edges selected uniformly at random.

We do this for a range of graph sizes. For each graph size, we perform the test 8 times. The results of this scaling test can be seen in Figure 4.3. Along the x-axis is the total size of the graph being tested. The red line shows the mean time to perform the decomposition to construct the solver, while the blue line shows mean time to perform a solve after the solver has been constructed.

The favorable scaling of this method is clear, as the plot appears nearly linear.

4.7.2 Preconditioning

In this section we explore the effectiveness of using our method as a preconditioner within the Preconditioned Conjugate Gradient method.

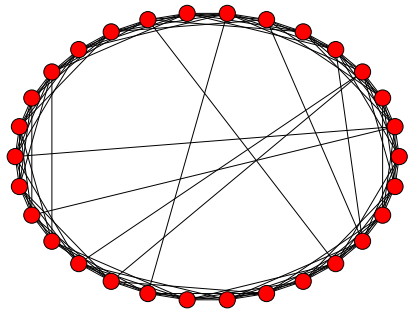
We have not, in this paper, specified methods for computing the recursive bisection on a graph or deciding which edges to keep between atomic subset in the support graph L_S. For 2D and 3D grids, we select the longest axis-aligned dimension and cut along its middle, spreading the support graph edges out evenly along that cutting plane.

For other graphs, we recursively bisect using METIS [35], constrained to give connected partitions, and we select edges to keep in L_S uniformly at random. We believe this to be a reasonable heuristic for guidelines 1 and 2 at the end of Section 4.6.

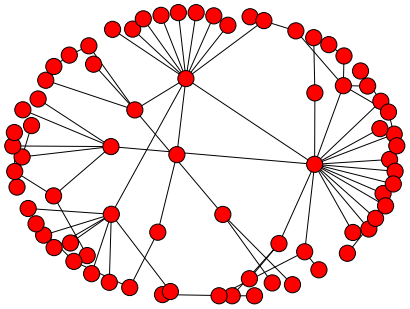
First, we test the behavior of our method on four different types of artificial graphs:

- Two-dimensional square grids.
- Three-dimensional cube grids.
- Watts-Strogatz random graph models [75].
- Barabási-Albert random graph models [10].

The Watts-Strogatz random graph model is a "ring lattice with random rewiring." It a popular random graph model used to capture behavior in which



(a) A sample Watts-Strogatz graph.



(b) A sample Barabási-Albert graph.

groups of nodes tend to be tightly clustered but still have some "long range" edges. An example can be seen in Figure 4.4(a).

The Barabási-Albert random graph model is a popular "preferential attachment" model in which nodes are added to the graph sequentially. New nodes are connected to other nodes at random, with the probability of connection proportional to the degree of that node. It captures the skewed degree distribution present in many social networks, though there is no clustering in this model. An example can be seen in Figure 4.4(b).

Although artificial graphs are different from real-world graphs, they are useful because they allow us to study the behavior of our method as the size of the problem grows on a given class of graphs. For each of the above four graph types, we examine rates of convergence while varying problem size, acceptable support graph cut size, and coarse graph size.

In the following, let n_A denote the maximum atomic subgraph size and let n_c denote the coarse graph size. Unless stated otherwise, the number of edges per cut in the support graph is also n_A.

Constant Cut Size, Linear Coarse Graph Size

Here we study the behavior of our method when the maximum atomic subgraph and support graph cut size are kept constant, but the coarse grid size n_c is allowed to grow linearly.

Figures 4.9 and 4.10 show the results of these tests on artificial graphs. These tests show that the coarse-grid correction drastically improves performance of PCG in most cases.

On grids, after an initial rise, the convergence rate of the two-grid method is independent of graph size. Note that some tests only require a single iteration because the cut size is large enough that $L_S = L$ in those cases.

On Watts-Strogatz graphs, we test with mean degree $k = 10$ and rewiring probability $\beta = 0.1$. The rate of growth of iteration count is largely independent of n_A, but increasing n_A tends to improve convergence by a constant.

On Barábasi-Albert graphs, we test with new node degree $m = 10$. The displayed regression lines are less informative here, especially for $n_A = 256$. However, we see that increasing n_A tends to improve convergence for smaller graphs, and slightly hamper convergence for larger graphs.

In Figure 4.4, we test on an array of networks from the Stanford Large Network Database, [41], and the 10th DIMACS Challenge, [9]. For a complete list of the networks tested, see Appendix 4.9.

As n_A grows, both the one-level and two-level methods improve, but the difference between the two shrink. This suggests that the increasing cut size tends to matter more than the decreasing coarse-grid size. This is different behavior

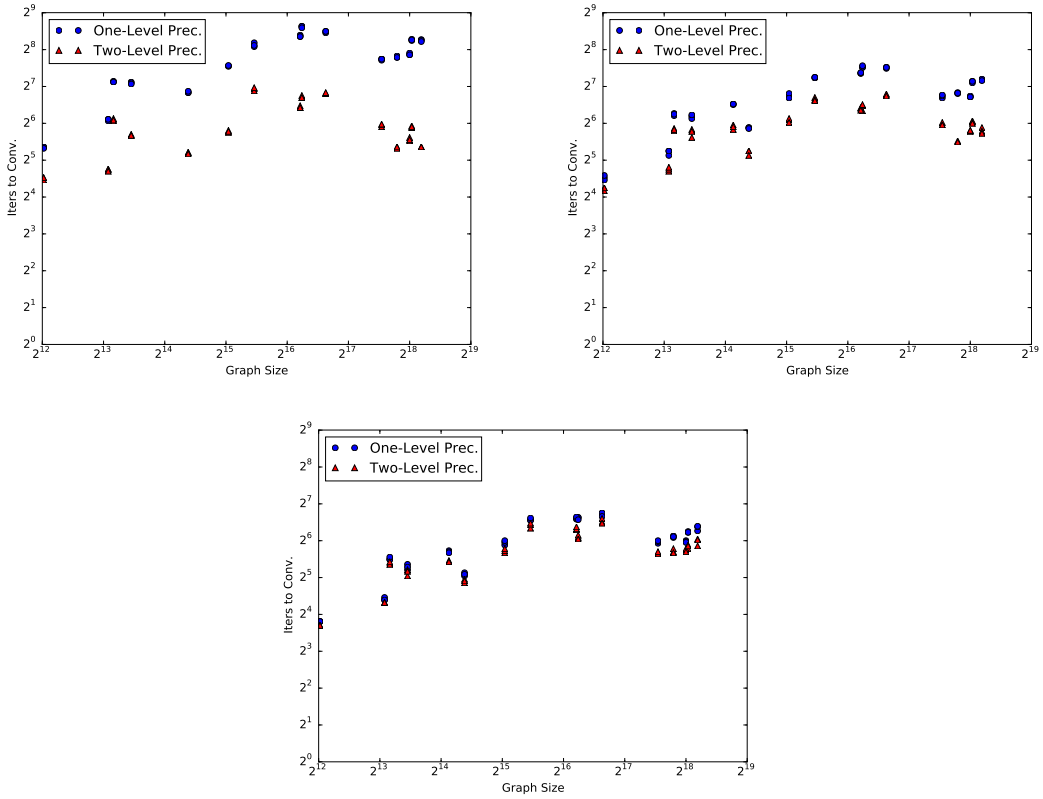


Figure 4.4: Iterations required to reach a residual norm of 10^{-6} and log-log regression lines for various real-world graphs obtained from SNAP and the 10th DIMACS Challenge. Maximum atomic subgraph size is 16, 64, and 256, respectively.

than seen in the Barábasi-Albert model; we hypothesize that this is due to the tendency of real-world networks to cluster, while Barábasi-Albert graphs have no clustering.

Increasing Cut Size, Constant Coarse Grid Size

In this section we examine purely the effect of increasing the acceptable cut size in the support graph without changing n_A or the size of the coarse graph. We study a 2D grid of size 320×320, a 3D grid of size $47 \times 47 \times 47$, a Watts-Strogatz

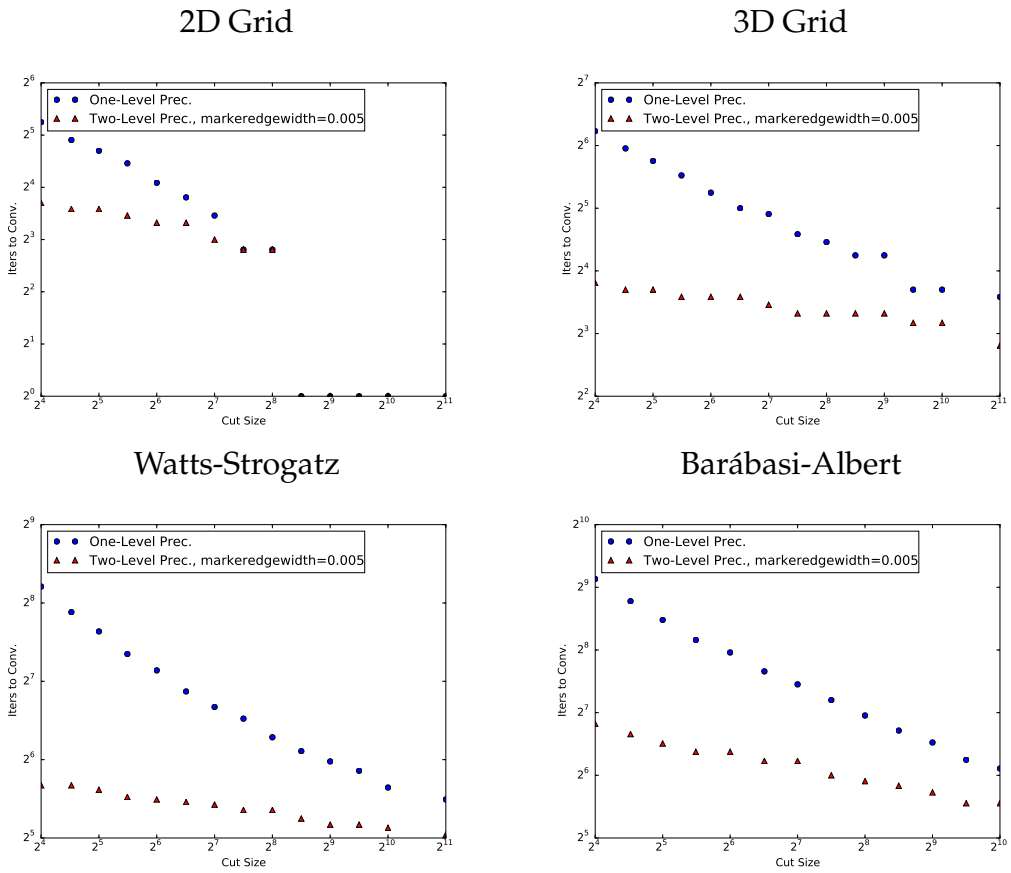


Figure 4.5: Maximum cut size in L_S vs. iterations required to reach a residual norm of 10^{-6} for several artificial graphs. All graphs have approximately 103,000 nodes. While both one-level and two-level methods improve with increasing cut size, the two-level method is significantly less sensitive.

graph of size 103,000, and a Barábasi-Albert graph also of size 103000. We use the same parameters for the random graph models as in the previous section. The results in Figure 4.5 show that while both methods improve with increasing acceptable cut size, the two-level method is significantly less sensitive to these changes.

We also tested increasing cut sizes on a selection of real-world networks. These results can be seen in Figure 4.6. Again we see that while increasing cut

size improves convergence, the two-level method is less sensitive to this change than the one-level method.

Constant Cut Size, Increasing Coarse Grid Size

In this section we examine purely the effects of shrinking coarse node size, or equivalently, increasing coarse grid size n_c, without changing n_A or the acceptable cut size. As in the previous section, we study a 2D grid of size 320×320, a 3D grid of size $47 \times 47 \times 47$, a Watts-Strogatz graph of size 103,000, and a Barábasi-Albert graph also of size 103000. We use the same parameters for the random graph models as above. Note that the size of a coarse grid scales approximately linearly with the inverse of the coarse node sizes. We choose sets of nodes to aggregate in the creation of a coarse graph as subsets V_i^ν obtained in the partition hierarchy.

The results in Figure 4.7 show that the rate of convergence is most improved on 2D grid, 3D grid, and Watts-Strogatz graphs for smaller coarse nodes, but that the Barábasi-Albert graphs still show significant improvement even for large coarse node sizes. However, the Barábasi-Albert graphs still require the most iterations.

We also tested shrinking coarse node size (increasing n_c) on a selection of real-world networks. These results can be seen in Figure 4.8. Most of the real-world graphs show significant improvements in convergence as the coarse nodes shrink, except for soc-Slashdot0811, which improves but not as drastically.

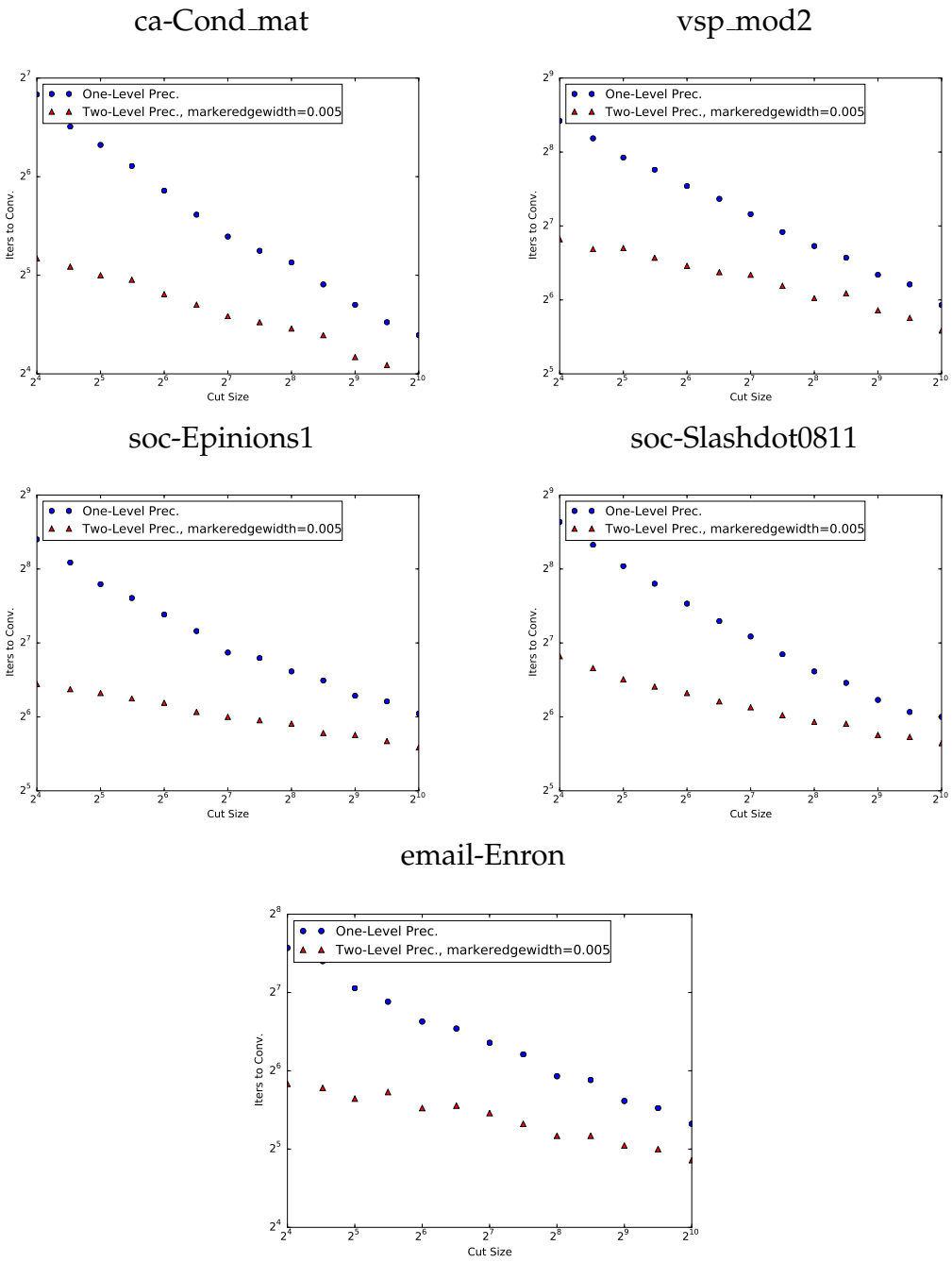


Figure 4.6: Maximum cut size in L_S vs. iterations required to reach a residual norm of 10^{-6} for several real-world networks. While both one-level and two-level methods improve with increasing cut size, the two-level method is less sensitive.

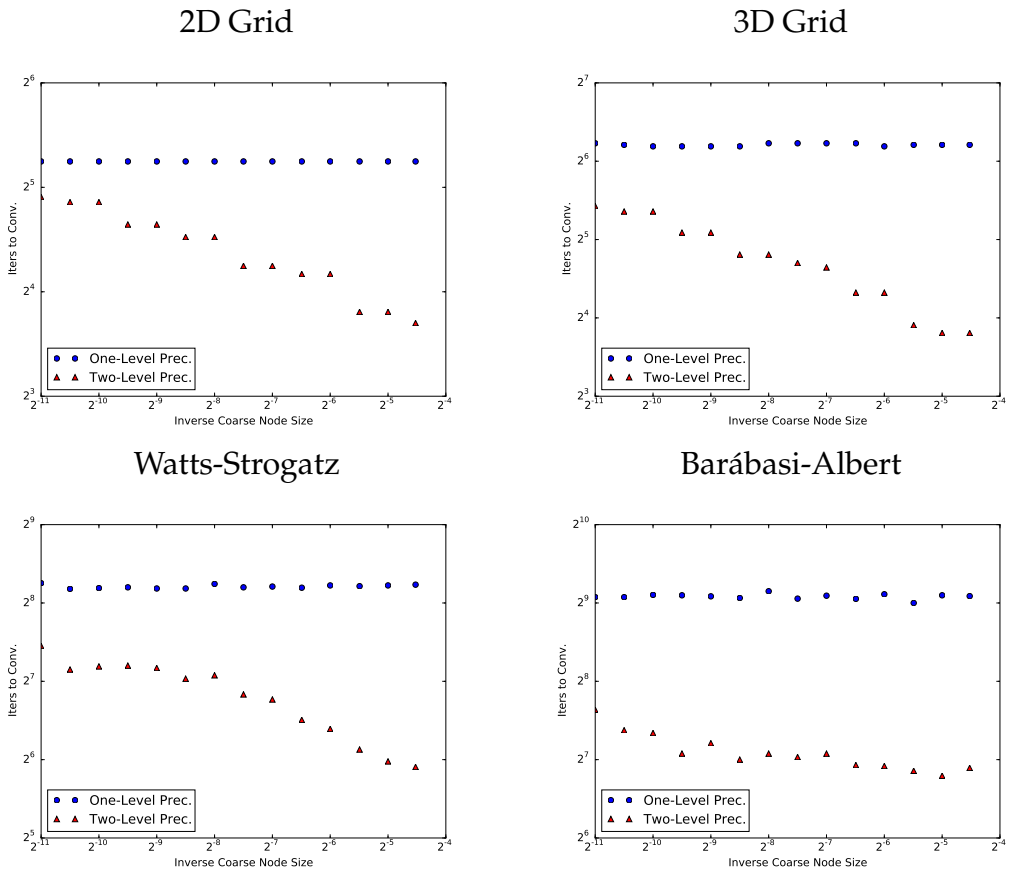


Figure 4.7: Inverse coarse node ($\propto n_c$) size vs. iterations required to reach a residual norm of 10^{-6} for several artificial graphs. All graphs have approximately 103,000 nodes. While most graph types show improved convergence with increasing coarse grid size (i.e. shrinking coarse node size), the Barábasi-Albert graphs show significantly improved convergence even for small coarse grids.

4.8 Conclusion

In this paper, we presented a parallelizable method for solving graph Laplacian-based linear systems derived from graphs that can be hierarchically bipartitioned with small edge cuts and showed that this method solves such systems in time $O(n \log n)$. We then used this method to construct a support graph-based preconditioner for graph Laplacian systems that do not have this property. Fi-

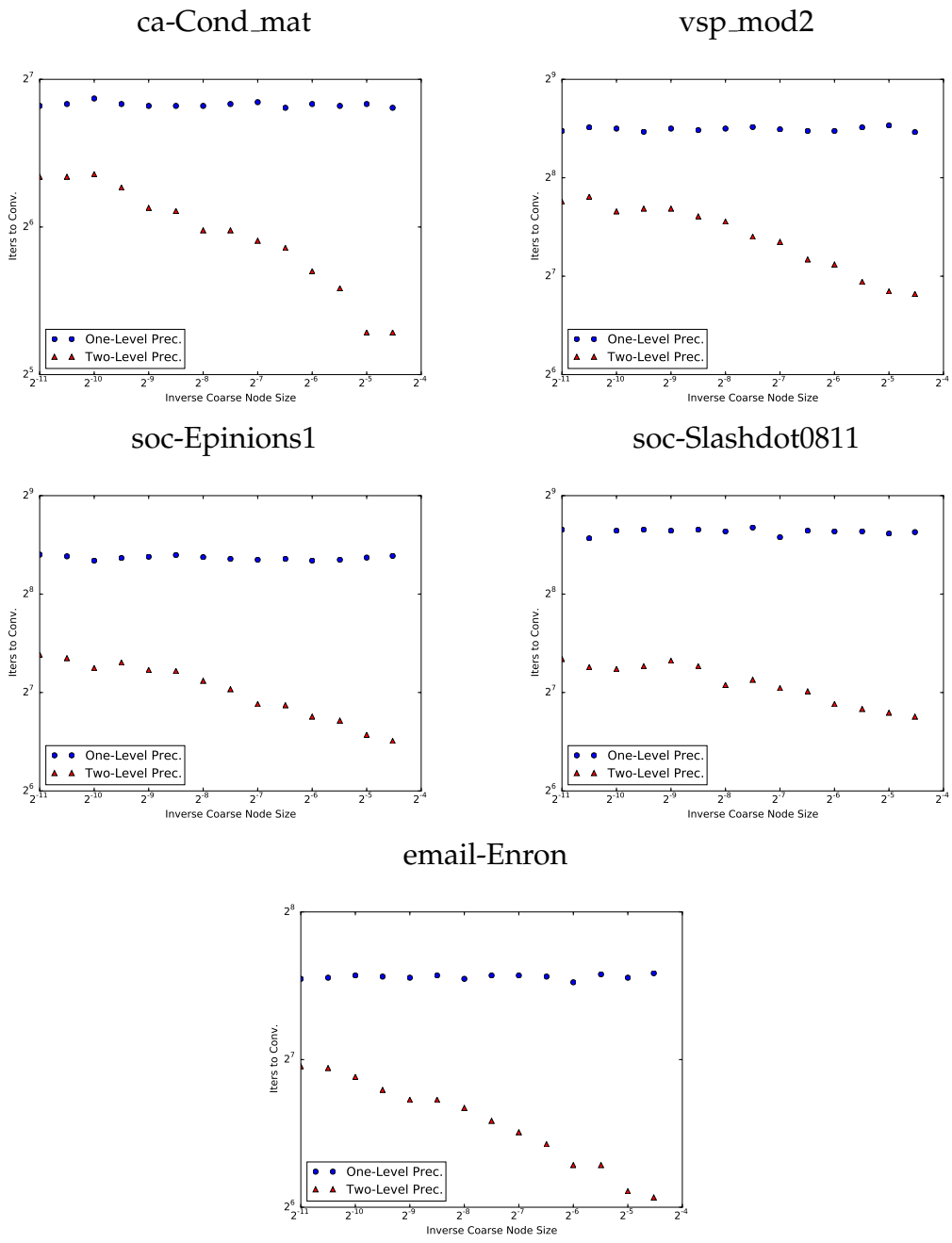


Figure 4.8: Inverse coarse node size vs. iterations required to reach a residual norm of 10^{-6} for several real-world networks. Some graphs behave like the Watts-Strogatz grahs of Figure 4.7, others behave like the Barábasi-Albert graphs.

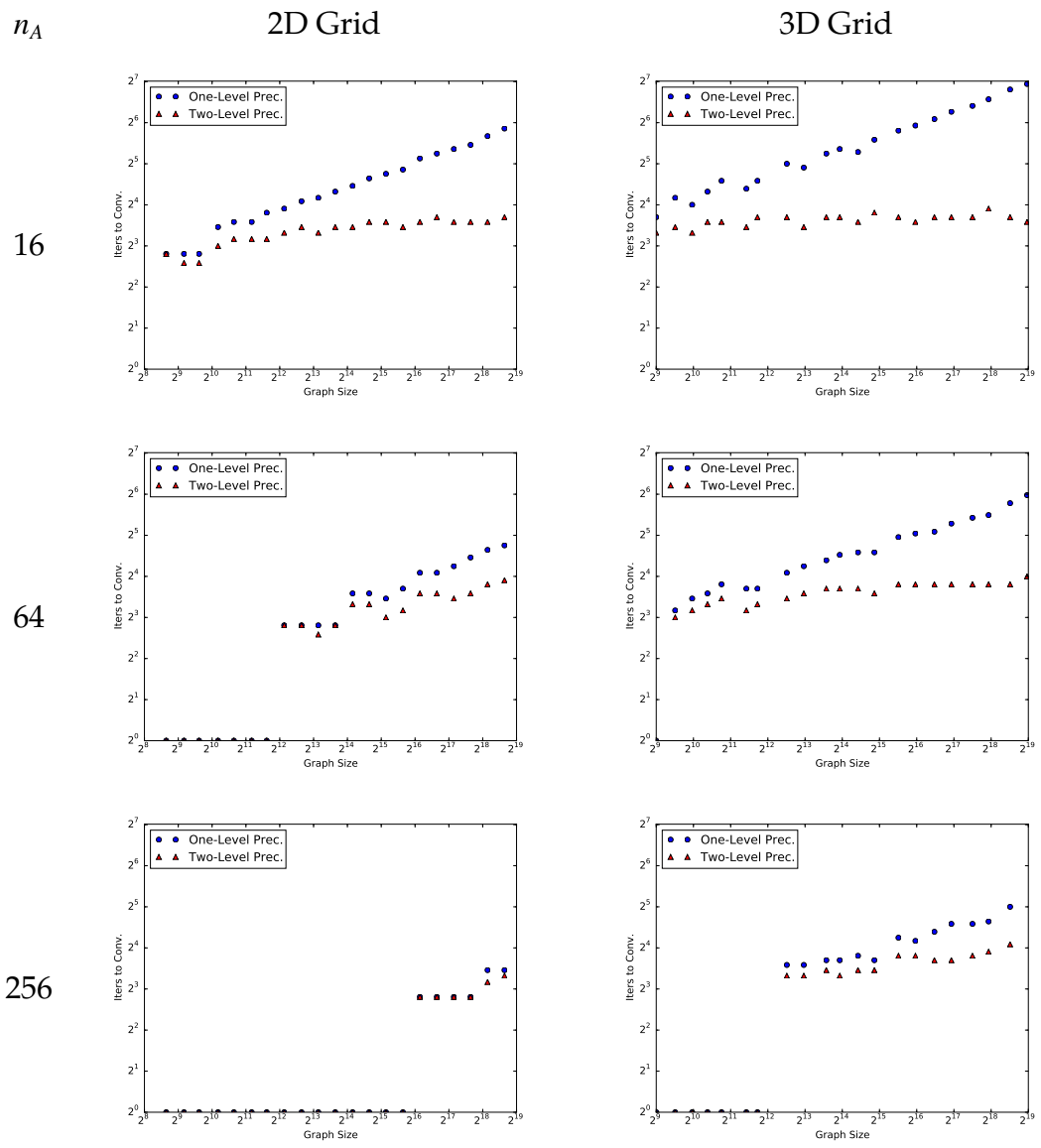


Figure 4.9: Iterations required to reach a residual norm of 10^{-6} for 2D and 3D grids using both our one-level and two-level preconditioners with maximum atomic subgraph size n_A, support graph cut size n_A, and coarse grid size $n_c \approx n/n_A$. After an initial rise, iteration count is independent of graph size.

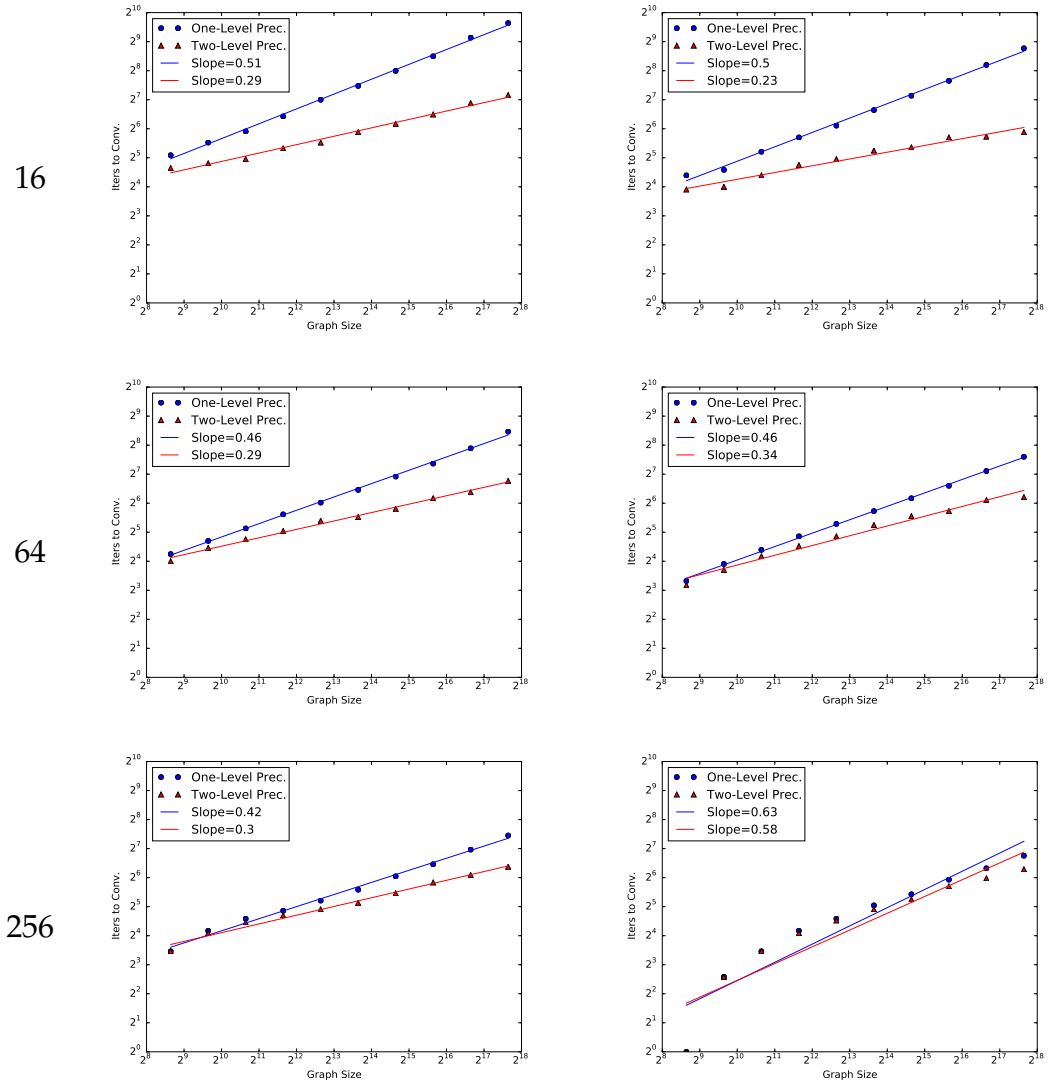


Figure 4.10: Iterations required to reach a residual norm of 10^{-6} for Watts-Strogatz and Barábasi-Albert graphs using both our one-level and two-level preconditioners with maximum atomic subgraph size n_A, support graph cut size n_A, and coarse grid size $n_c \approx n/n_A$. Two-grid performance on larger graphs is robust to variations in n_A.

nally, we augmented this method with a two-grid approach to account for the weaknesses in the one-level preconditioner. We presented an analysis of the two-grid method, as well as a theorem deriving the condition number of two-grid support graph-based preconditioners.

We did not develop methods for partitioning a graph, or for selection of edges to keep in a support graph. Future work will include addressing these issues, especially in the context of graph Laplacians derived from real-world networks. In addition, we will explore the use of recursive coarse-grid corrections in a multigrid method.

4.9 Appendix: List of Real-World Graphs Tested

- **SNAP Networks**
  - ca-CondMat
  - ca-GrQc
  - soc-Epinions1
  - soc-Slashdot0811
  - amazon0302
  - ca-HepTh
  - ca-HepPh
  - email-Enron
- **10th DIMACS Networks**
  - citationCiteseer
  - caidaRouterLevel
  - coAuthorsCiteseer
  - coAuthorsDBLP

CHAPTER 5

CONCLUSION

In Chapter 1, we introduced the concept of error flattening, in which two complementary solver techniques are used together: one that simplifies the error of a problem, typically so that is lies in or near a low-dimensional subspace; and one that (partially or completely) solves problems whose errors lie within that subspace. We also discussed both matrix augmentation-based solves and coarse-grid projections as examples of error flattening. Each of the following chapters used at least one of these two error flattening approaches to create efficient solvers for network-structured problems.

In Chapter 2, we used small matrix augmentation, and in some cases the Sherman-Morrison-Woodbury formula (see Section 1.3.1), to accelerate the process of identifying topology changes in power networks. It is of note that although matrix augmentation allowed us to accelerate the solution process, the more drastic speed improvement came from the filtering procedure. Recall that in this case the goal was not truly to find the solutions to the linear systems, but to identify the topology change among a finite set of possibilities. For each possibility, the first step of error flattening was cheap and reduced the error to a *known* low-dimensional subspace. Before even applying the second step of error flattening, comparing this subspace to the observed voltage fingerprint allowed us to drastically reduce the number of reasonable possibilities that we needed to check.

Although our method is quite accurate, transmission and distribution network operators require extremely high accuracy in order to be comfortable using a technique on a live power grid. Future work in this area should focus on

improving the accuracy of our method while maintaining as much efficiency as possible. In addition, the current requirement that the network settle to a steady state limits this method's use in potentially unstable situations. An extension of this method to the time domain could make it useful in that case.

In Chapter 3, we used coarse-grid projections to develop a nonlinear algebraic multigrid approach to solving the power flow equations. Most multigrid research focuses on linear problems, and most of the nonlinear methods require an underlying geometry such as one has with partial differential equations [14]. Here, we present a nonlinear multigrid method that does not require knowledge of an underlying geometry, but is truly algebraic in nature. In addition, like most multigrid methods, our approach is highly scalable to large problem sizes, which makes our method potentially important to power engineers who wish to simulate very large networks.

While we have demonstrated the scalable nature of our approach, we have yet to compare it against existing commercial solvers. Future work includes building a high-performance implementation of our method for this comparison. One interesting possibility is that our method might be best used *with* existing commercial methods rather than *instead* of them: current commercial solvers are extremely fast at solving small and medium sized problems. Our experiments show that one step of coarsening with a direct solve on the coarse grid converges quite quickly, so it could be highly effective to use a commercial solver on that coarse problem.

Finally, in Chapter 4, we used both the Sherman-Morrison-Woodbury formula and coarse-grid projections to develop a fast solver for graph Laplacians based on natural networks that have too much community structure for a sim-

ple Jacobi iteration but not enough separability for Nested Dissection [43]. Here, we combine two well-known solution approaches: graph Laplacian preconditioners based on support graphs that are spanning trees plus a small number of extra edges [63], and multigrid methods [14]. While our support graph preconditioner is not as theoretically rigorous as those of [63], it uses existing software and is fast to apply. The result is a solution method that effective on this difficult class of graph Laplacians.

This method currently creates its support graph by recursively bisecting with METIS [35] and removing edges uniformly at random until an it obtains an acceptable cut size. Future work should focus on improving this procedure, perhaps by narrowing the class of graphs under consideration. Importantly, in this chapter we developed a proof of rates of convergence for two-grid support graph preconditioners, and this will act as a guideline in further research. Future work should also address how best to scale to more hierarchy levels than two, as this is more challenging with support graph preconditioners, and is an important issue for extremely large graph Laplacians.

We have demonstrated here the power and broad applicability of error flattening. In some cases the two complementary error flattening steps are clear, as in the case of Chapter 2. In other cases, such as in Chapters 3 and 4, one must design approximate solvers for each step that one uses in an iterative procedure. In both cases, error flattening techniques can result in highly effective solution methods for otherwise expensive problems.

BIBLIOGRAPHY

[1] Number of monthly active facebook users worldwide as of 3rd quarter 2015 (in millions). http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/. Accessed: 2015-12-01.

[2] A Abur, Hongrae Kim, and MK Celik. Identifying the unknown circuit breaker statuses in power networks. *IEEE Transactions on Power Systems*, 10(4):2029–2037, 1995.

[3] V. Ajjarapu and C. Christy. The continuation power flow: a tool for steady state voltage stability analysis. *IEEE Power and Energy Society*, 7(1):416–423, 1992.

[4] O. Alsac, B. Stott, and W. F. Tinney. Sparsity-oriented compensatin methods for modified network solutions. *IEEE Trans. Power Apparatus and Syst.*, PAS-102(5):1050–1059, May 1983.

[5] O. Alsaç, N. Vempati, B. Stott, and A. Monticelli. Generalized state estimation. *IEEE Transactions on Power Systems*, 13(3):1069–1075, 1998.

[6] A. Ashok and M. Govindarasu. Cyber attacks on power system state estimation through topology errors. In *IEEE PES Meeting*, pages 1–8, 2012.

[7] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, 1985.

[8] F.A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. Jacob Filho, R. Lent, and S. Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, 2009.

[9] D. A. Bader, A. Kappes, H. Meyerhenke, P. Sanders, Schulz C., and Wagner D. Benchmarking for graph clustering and partitioning. In R. Alhajj and J. Rokne, editors, *Encyclopedia of Social Network Analysis and Mining*. Springer-Verlag, New York, 2014.

[10] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[11] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31:333–390, 1977.

[12] A. Brandt, S. F. McCormick, and J. W. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In David J. Evans, editor, *Sparsity and its Application*, pages 257–284, Cambridge, 1985. Cambridge University Press.

[13] M. Brezina, P. Vaněk, and P. S. Vassilevski. An improved convergence analysis of smoothed aggregation algebraic multigrid. *Numerical Linear Algebra with Applications*, 19(3):441–469, 2012.

[14] W. L. Briggs, V. E. Hensen, and S. F. McCormick. *A Multigrid Tutorial, 2nd Ed.* SIAM, Philadelphia, 2000.

[15] E. Caro, A. J. Conejo, and A. Abur. Breaker status identification. *IEEE Transactions on Power Systems*, 25(2):694–702, 2010.

[16] C. Chevalier and F. Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel Matrix Algorithm and Applications*, 34(6–8):318–331, 2008.

[17] K. A. Clements and A. S. Costa. Topology error identification using normalized Lagrange multipliers. *IEEE Transactions on Power Systems*, 13(2):347–353, 1998.

[18] K.A. Clements and P.W. Davis. Detection and identification of topology errors in electric power systems. *IEEE Transactions on Power Systems*, 3(4):1748–1753, 1988.

[19] IS Costa and JA Leao. Identification of topology errors in power system state estimation. *IEEE Transactions on Power Systems*, 8(4):1531–1538, 1993.

[20] T. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, 2006.

[21] Antonio De La Villa Jaén and Antonio Gómez-Expósito. Implicitly constrained substation model for state estimation. *IEEE Transactions on Power Systems*, 17(3):850–856, 2002.

[22] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[23] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, Providence, RI, 1998.

[24] R. D. Falgout and J. B. Schroder. Non-galerkin coarse grids for algebraic multigrid. *SIAM Journal on Scientific Computing*, 36(3):309–334, 2014.

[25] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov. On two-grid convergence estimates. *Numerical Linear Algebra With Applications*, 12(5–6):471–494, 2005.

[26] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov. On two-grid convergence estimates. *Numerical Linear Algebra Appl.*, 12:471–494, 2005.

[27] D. G. Feingold and R. S. Varga. Block diagonally dominant matrices and generalizations of the gershgorin circle theorem. *Pacific Journal of Mathematics*, 12(4):1241–1250, 1962.

[28] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[29] A. George. Nested dissection of a regular finite element mesh. *Siam Journal on Numerical Analysis*, 10(2):345–363, 1973.

[30] J. D. Glover, M. S. Sarma, and T. J. Overbye. *Power System Analysis and Design*. Cengage Learning, Stamford, 2012.

[31] A Gómez Expósito and Antonio de la Villa Jaén. Reduced substation models for generalized state estimation. *IEEE Transactions on Power Systems*, 16(4):839–846, 2001.

[32] W. W. Hager. Updating the inverse of a matrix. *SIAM Rev.*, 31(2):221–239, June 1989.

[33] R. Idema, G. Papaefthymiou, D. Lahaye, C. Vuik, and L. van der Sluis. Towards faster solution of large power flow problems. *IEEE Transaction on Power Systems*, 28(4):4918–4925, 2013.

[34] M. R. Irving and M. J. H. Sterling. Substation data validation. In *IEE Proc. C*, volume 129, pages 119–122. IET, 1982.

[35] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(2):359–392, 1999.

[36] S. K. Khaitan and J. D. McCalley. A class of new preconditioners for linear solvers used in power system time-domain simulation. *IEEE Power & Energy Society*, 25(4):1835–1844, 2010.

[37] S. K. Khaitan, J. D. McCalley, and Q. Chen. Multifrontal solver for online power system time-domain simulation. *IEEE Transactions on Power Systems*, 23(4):1727–1737, 2008.

[38] A. Knyazev and K. Neymeyr. Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method. *Electronic Transactions on Numerical Analysis*, 15:38–55, 2003.

[39] D. P. Kothari and I. J. Nagrath. *Power System Engineering*. Tata McGraw-Hill, West Patel Nagar, New Delhi, 2008.

[40] I. Koutis, G. L. Miller, and R. Peng. A nearly-m log n time solver for sdd linear systems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598, 2011.

[41] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[42] X. Li, F. Li, and J. M. Clark. Exploration of multifrontal method with gpu in power flow computation. In *Power and Energy Society General Meeting (PES)*, pages 1–5. IEEE, 2013.

[43] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(3):346–358, 1979.

[44] E.M. Lourenco, A. J. A. Costa, K. A. Clements, and R. A. Cernev. A topology error identification method directly based on collinearity tests. *IEEE Transactions on Power Systems*, 21(4):1920–1929, 2006.

[45] C. N. Lu, J. H. Teng, and W.-H. E. Liu. Distribution system state estimation. *IEEE Transactions on Power Systems*, 10(1):229–240, 1995.

[46] RL Lugtu, DF Hackett, KC Liu, and DD Might. Power system state estimation: Detection of topological errors. *IEEE Trans. Power Apparatus and Syst.*, PAS-99(6):2406–2412, 1980.

[47] L Mili, G Steeno, F Dobraca, and D French. A robust estimation method for topology error identification. *IEEE Transactions on Power Systems*, 14(4):1469–1476, 1999.

[48] A. Monticelli. Modeling circuit breakers in weighted least squares state estimation. *IEEE Transactions on Power Systems*, 8(3):1143–1149, 1993.

[49] A. Monticelli. Electric power system state estimation. *Proc. IEEE*, 88(2):262–282, 2000.

[50] A Monticelli and A Garcia. Modeling zero impedance branches in power system state estimation. *IEEE Transactions on Power Systems*, 6(4):1561–1570, 1991.

[51] J. L. Moreno. *Who Shall Survive? Foundations of Sociometry, Group Psychotherapy, and Sociodrama*. Beacon House Inc., Beacon, NY, 1953.

[52] S. Naka, T. Genji, T. Yura, and Y. Fukuyama. A hybrid particle swarm optimization for distribution state estimation. *IEEE Transactions on Power Systems*, 18(1):60–68, 2003.

[53] Taher Niknam and Bahman Bahmani Firouzi. A practical algorithm for distribution state estimation including renewable energy sources. *Renewable Energy*, 34(11):2309–2316, 2009.

[54] Y. Notay. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis*, 37:123–146, 2010.

[55] A. G. Phadke and J. S. Thorp. *Synchronized Phasor Measurements and Their Applications*. Springer, New York, 2008.

[56] R. Podmore. Coherency in power systems. In J. H. Chow, editor, *Power System Coherency and Model Reduction*. Springer Science, New York, 2013.

[57] Katherine M Rogers, Rebecca D Spadoni, and Thomas J Overbye. Identification of power system topology from synchrophasor data. In *IEEE/PES Power Systems Conference and Exposition (PSCE)*, pages 1–8. IEEE, 2011.

[58] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 2003.

[59] G. Sánchez-Ayala, J. R. Agüero, D. Elizondo, and M. Lelic. Current trends on applications of PMUs in distribution systems. In *2013 Innovative Smart Grid Technologies*, pages 1–6, 2013.

[60] J. Sexauer, P. Javanbakht, and S. Mohagheghi. Phasor measurement units for the distribution grid: Necessity and benefits. In *2013 Innovative Smart Grid Technologies*, pages 1–6, 2013.

[61] Harmohan Singh and Fernando L Alvarado. Network topology determination using least absolute value state estimation. *IEEE Transactions on Power Systems*, 10(3):1159–1165, 1995.

[62] S. Smith, C. Woodward, L. Min, C. Jing, and A. Del Rosso. On-line transient stability analysis using high performance computing. In *Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5. IEEE, 2014.

[63] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *36th Annual ACM Symposium on Theory of Computing*, pages 81–90, 2004.

[64] B. Stott. Review of load-flow calculation methods. *Proceedings of the IEEE*, 62(7):916–929, 1974.

[65] B. Stott. Power system dynamic response calculations. *Proceedings of the IEEE*, 67(2):219–241, 1979.

[66] B. Stott and O. Alsaç. Fast decoupled load flow. *IEEE Transaction on Power Apparatus and Systems*, 93(3):859–869, 1974.

[67] Joseph Euzebe Tate and Thomas J Overbye. Line outage detection using phasor angle measurements. *IEEE Transactions on Power Systems*, 23(4):1644–1652, 2008.

[68] Joseph Euzebe Tate and Thomas J Overbye. Double line outage detection using phasor angle measurements. In *IEEE PES Meeting*, pages 1–5. IEEE, 2009.

[69] W. F. Tinney and C. E. Hart. Power flow solution by newton's method. *IEEE Transaction on Power Apparatus and Systems*, 86(11):1449–1460, 1967.

[70] U.S.-Canada Power System Outage Task Force. Final report on the implementation of the task force recommendation. Available `http://energy.gov/oe/downloads/blackout-2003-blackout-final-implementation-report`, September 2006.

[71] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[72] P. Vassilevski and L. Zikatanov. Commuting projectsion on graphs. *Numerical Linear Algebra with Applications*, 21(3):297–315, 2013.

[73] P. S. Vassilevski. *Multilevel Block Factorization Preconditioners*. Springer, Livermore, California, 2008.

[74] F. Vosgerau, I. S. Costa, K. A. Clements, and E. M. Lourenco. Power system state and topology coestimation. In *Bulk Power System Dynamics and Control (iREP) - VIII*, pages 1–6, 2010.

[75] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[76] Felix F Wu and W.-H. E. Liu. Detection of topology errors by state estimation. *IEEE Transactions on Power Systems*, 4(1):176–183, 1989.

[77] Tao Yang, Hongbin Sun, and Anjan Bose. Transition to a two-level linear state estimator - part II: Algorithm. *IEEE Transactions on Power Systems*, 26(1):54–62, 2011.

[78] Tao Yang, Hongbin Sun, and Anjan Bose. Transition to a two-level linear state estimator - part I: Architecture. *IEEE Transactions on Power Systems*, 26(1):46–53, 2011.

[79] Y.-S. Zhang and H.-D. Chiang. Fast newton-fgmres solver for large-scale power flow study. *IEEE Transactions on Power Systems*, 25(2):769–776, 2010.

[80] Hao Zhu and Georgios B Giannakis. Lassoing line outages in the smart power grid. In *SmargGridComm*, pages 570–575. IEEE, 2011.

[81] Hao Zhu and Georgios B Giannakis. Sparse overcomplete representations

for efficient identification of power line outages. *IEEE Transactions on Power Systems*, 27(4):2215–2224, 2012.

[82] R. D. Zimmerman, C. E. Murillo-Sánchez, , and R. J. Thomas. MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.